GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, Exploits, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

GIAC IHHE Practical Assignment:

BIND 8.2 NXT Remote Buffer Overflow Exploit

Prepared for:
SANS Global Incident Analysis Center

By:
Robert McMahon
rwm@mcmahoncpa.com

Aug 13, 2000

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

1. Exploit Details 2

2. Protocol Description 2

3. Description of Variants 3

4. How the Exploit Works 3

5. How to Employ the Exploit 6

6. Attack Signature 8

7. How to Protect Against the Attack 9

8. Source Code/Pseudo Code 10

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

1. Exploit Details

- Name: BIND 8.2 NXT remote buffer overflow exploit
- CVE Number: CVE-1999-0833
- CERT Advisories:
  - http://www.cert.org/advisories/CA-2000-03.html
  - http://www.cert.org/advisories/CA-99-14-bind.html
- Operating System: Systems running BIND 8.2, 8.2.1 with Linux, Solaris, FreeBSD, OpenBSD, and NetBSD Unix operating systems. Prior versions of BIND, including 4.x, are not vulnerable to this particular exploit.
- Protocols/Services: TCP/UDP, port 53
- Description: The early versions of BIND that introduced the NXT resource record extension improperly validated these records inputs. This " bug" permits a remote attacker to execute a buffer overflow in order to gain access to a target system at the same privilege level the *named* daemon is running at, e.g., root.

2. Protocol Description

The Domain Name System (DNS) is one of the most widespread protocols utilized on the Internet because of its function - resolving domain names to IP addresses. Email messaging and web browsing would be at best chaotic if DNS was denied to public use. DNS is based on a client-server distributed architecture composed of resolvers and name servers. Name servers that perform *recursive* resolution (as apposed to *iterative* resolution) are of particular interest because of their vulnerable to the NXT remote exploit on certain DNS implementations.[1]

DNS uses both UDP and TCP transport protocols. Resolvers and name servers query other name servers using UDP, port 53 for almost all standard queries. TCP is used for zone transfers and also for queries of "larger size" domain names (e.g., exceeding 512 Bytes), which has relevance to the subject exploit.

Earlier versions of DNS were regarded as insecure since there was no ability to authenticate name servers. In an attempt to make this protocol more secure and permit authentication, DNS Security Extensions were developed. One of these extensions is the *NXT* Resource Record (RR). The NXT RR provides the ability to "securely" deny the existence of a queried resource record *owner name* and *type*. Ironically, it is this security feature that opens the door for the subject buffer overflow attack and is the reason why earlier versions of BIND were not exposed. The details of the NXT Resource Record and associated data fields can be found in RFC 2065, [http://www.freesoft.org/CIE/RFC/2065/index.htm].

[1]See *DNS and BIND, 2nd Edition* by Paul Albitz and Cricket Liu, O'Reilly & Associates, Inc., for a detailed description on how DNS works.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

The BIND (Berkeley Internet Name Domain) implementation of DNS is the most popular version deployed on the Internet. The BIND 8.2 implementation of the NXT RR was developed with a programming bug in it that permits remote intruders (via another name server) to execute arbitrary code with the privileges of the user running the *named* daemon. The specifics on this programming bug are discussed in paragraph 4 below.

3. Description of Variants

The version of the NXT exploit addressed in this paper was written by Horizon and Plaguez of the *ADM CreW* [ftp://freelsd.net/pub/ADM/exploits/t666.c]. This version has successfully engaged several name servers.

Another version of the NXT remote exploit, *Exploit for BIND-8.2/8.2.1 (NXT),* was written by the *TESO* group [http://teso.scene.at/releases.php3/teso-nxt.tar.gz]. Because the author "z-" gives thanks to Horizon, it is assumed this code was developed after the ADM-NXT version.[2] Some key differences that were noticed with a cursory examination, other than the differences due to programming style, were the following.

- The ADM-NXT version was tampered with by the authors to make it harder for "script kiddies" to employ.
- The TESO-NXT version was only designed to run against Linux and FreeBSD operating systems memory stacks.

4. How the Exploit Works

The BIND 8.2 NXT exploit is based on a buffer overflow of the stack memory. This buffer overflow is possible because of insecure coding practices. Many programmers employ functions that use routines that do not check the bounds of input variables. The reasons for this may be intentional (e.g., for performance reasons) or possibly just lacks of understanding of secure programming techniques. At any rate, this is an all too common practice and can be exploited by a hacker who has access to source code and can run utilities like *strings* that find insecure routines. (An understanding of C and assembly programming languages along with lots of patience would also be helpful.)

Of particular relevance to the BIND 8.2 NXT exploit as well as other buffer overflow attacks, is stack memory manipulation. Stack memory is the type of memory that programs use to store function local variables and parameters. An important concept regarding stack memory exploitation is related to the return pointer. The return pointer contains the address of the place in the calling program control is returned to after completion of the function[3]. The following example is a simple illustration describing how a buffer overflow attack can be used to overwrite the return pointer while inserting executable machine code in the stack.

[2] Based on research performed 12-13 August 2000, it could not be ascertained if the TESO-NXT version has ever been successfully deployed.
[3] *Advanced Incident Handling and Hacker Exploits: Step-by-Step, Part 1 and 2*, by Edward Skoudis, pg 183, SANS, July 5-10, 2000.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Example 1: A given function has two variables defined, var1[20] and var2[12]. For simplicity, the variables and the return point, ptr[0], are allocated addresses 00000000 through 00000020 as depicted in the figure 1.

Figure 1: Before Exploit

Addresses	Contents
00000000	STACK[0] = var1
00000004	STACK[1] = var1
00000008	STACK[2] = var1
0000000C	STACK[3] = var1
00000010	STACK[4] = var1
00000014	STACK[5] = var2
00000018	STACK[6] = var2
0000001C	STACK[7] = var2
00000020	**STACK[8] = PTR[0]**
00000024	STACK [9]
00000028	STACK [10]

To demonstrate the buffer overflow, if an input var2 exceeding 12 characters is entered into the stack via a routine such as *strcpy( )*, the return pointer, PTR[0] is overwritten by the overflow data of *var2** (see figure 2 below). In this case, *var2** was carefully crafted to be another return pointer that directs the flow of the function to address 00000024. This new address is the location of the attack payload, which was delivered as part of the overflow code. The payload can be machine executable code, such as */bin/sh –c*, which will run at the same privilege level of the program being exploited.

This example is a very simple demonstration on how a stack overflow can take control of a function to execute hacker-defined executable code. In reality it can be quite an endeavor to build a buffer overflow program, especially if the programmer has to predict the specific arrangements of the stack. One of the best resources available that describes stack memory concepts is the article "Smashing the Stack for Fun and Profit" by Aleph One, in *Phrack Magazine*, Issue 49, Article 14, [http://phrack.infonexus.com/search.phtml?view&article=p49-14].

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Figure 2: After Exploit

Addresses	Contents
000000000	STACK[0] = var1
000000004	STACK[1] = var1
000000008	STACK[2] = var1
00000000C	STACK[3] = var1
000000010	STACK[4] = var1
000000014	STACK[5] = var2
000000018	STACK[6] = var2
00000001C	STACK[7] = var2
000000020	**STACK[8] = var2***
000000024	Attack_ Payload[1]
000000028	Attack_Payload[2]

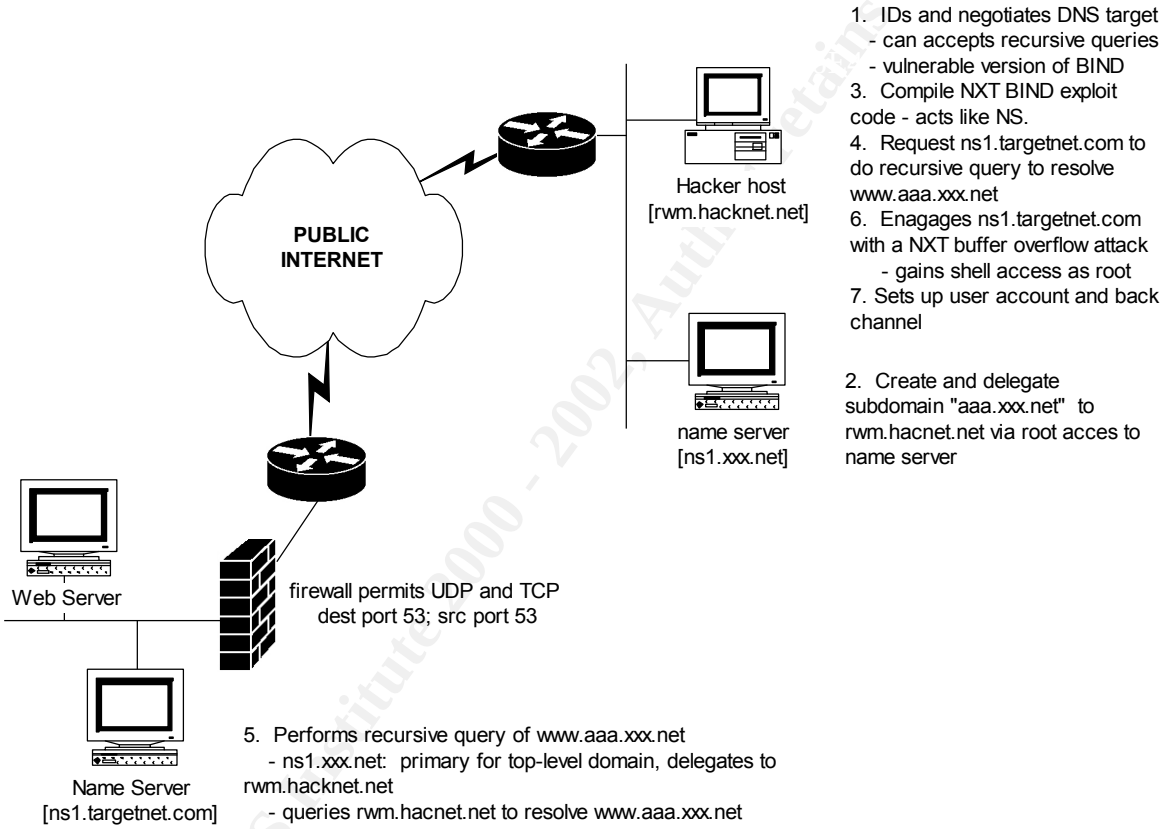
THE ADM-NXT BIND buffer overflow exploit works when the target name server performs a recursive DNS query on a hacker host. The query basically fetches a maliciously-constructed NXT record which contains the code that exploits the BIND server memory stack. The exploit code can be successfully engaged against primary, secondary, and even caching-only name servers. The next paragraph explains in more detail how the attack is actually employed.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

5. How to Employ the Exploit

The BIND 8.2 NXT remote buffer overflow exploit can be performed by a single machine, however, for purposes of providing a clear understanding of the host functions, the participating name server and hacker host (with NXT exploit code) will be denoted as separate machines (see figure 3 below).

Figure 3: BIND 8.2 NXT Remote Exploit Geometry



Step 1. Hacker host (rwm.hackernet.net) identifies and negotiates target name server.

- Determine if target name serve, ns1.targetnet.com, is vulnerable to NXT exploit via *dig* or *nslookup.* Like most firewall configurations on the Internet, the targetnet firewall permits DNS queries to UDP and TCP ports 53 from “any” host.
- Set up a resolver (/etc/resolv.conf) on rwm.hackernet.net to query ns1.xxx.net for it name services.
- Perform DNS queries of ns1.target.com in order to determine if it takes on burden of performing name queries – if so, then it performs recursive queries (e.g., name server does not just refer the requesting name server to different name server like it would for iterative query.)

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Step 2: Create and delegate subdomain

- Create following records on ns1.xxx.net:

aaa.xxx.net NS A rwm.hackernet.net
rwm.hackernet.net IN A 10.233.131.222

- Reinitialize in.named daemon…kill –HUP <in.named pid>

Step 3: Compile BIND 8.2 NXT exploit code (ADM-NXT version: t666.c)[4]

- Edit source code to change /adm/sh to /bin/sh (in hex) by searching the source code for 0x2f,0x61,0x64,0x6d,0x2f and replacing it with 0x2f,0x62,0x69,0x6e,0x2f. (The authors of the program, to put it in their words, wanted to raise the bar a little to make it harder for script kiddies to blindly execute this code.)
- Compile the *t666.c* source code with gnu C compiler and execute the bind_nxt executable

```
rwm #/tmp gcc t666.c -o bind_nxt
rwm #/tmp ./bind_nxt
```

Step 4: Request ns1.targetnet.com to do recursive query in order to resolve www.aaa.xxx.net - a host with subdomain delegated to rwm.hackernet.net as per NS record.

```
rwm #nslookup
> server ns1.targetnet.com
> www.aaa.xxx.net
```

Step 5: Target NS performs recursive queries to resolve www.aaa.xxx.net

- Queries ns1.xxx.net first since it is primary for top-level domain xxx.net. Receives message from ns1.xxx.net to query rwm.hacknet.net that is primary for subdomain aaa.xxx.net as per NS record.
- Queries rwm.hacnet.net to resolve www.aaa.xxx.net
- It should be noted that ns1.targetnet.net is running *in. named* with UID = 0

[4] From the article, *BIND 8.2 - 8.2.2 Remote root Exploit How-To* by E-Mind, [http://www.hack.co.za/daem0n/named/NXT-Howto.txt].

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Step 6: rwm.hacnet.net engages ns1.targetnet.com with a NXT buffer overflow attack

- rwm.hacnet.net sends a large NXT record containing code that exploits the remote BIND server memory stack with a buffer overflow (will use TCP instead of UDP because of the size of the transaction.)
- Hacker on rwm.hacnet.net gains shell access with privileges as root since in.named on target was running as root.

Step 7. Sets up user account and back channel

- Set up user account and backdoor (e.g., netcat listener) before exiting shell account (since buffer overflow caused DNS to crash).
- Come back and set up favorite rootkit.

6. Attack Signature

There are a number of signatures that the BIND 8.2 NXT remote buffer overflow (ADM-NXT) has. In many of the signatures, the two authors of the exploit source code, Horizon and Plaguez, deliberately leave their "signature" in various portions of the character array definitions portion. The ASCII and HEX versions of the code shown below can be easily retrieved by promiscuous-mode packet analyzers such as tcpdump, Snort, and Solaris' Snoop. With regard to the seven signatures listed, there is a strong likelihood more exist.

Signature 1: The recursive query request of a domain name that is not associated with the domain name of the server being queried. This could possibly be explained by a mistake in typing the domain name in the DNS query. However, it is assessed that this probability would become exponentially lower for domain names with characters exceeding four.

Signature 2: Some of the compromised systems had one of the following empty directories on systems where the NXT record vulnerability was successfully exploited [http://www.cert.org/advisories/CA-2000-03.html]:

/var/named/ADMROCKS
/var/named/O

Signature 3: On the BSD code version of the exploit, an empty file is created. The following came from the "char bsdcode[ ]=" portion of the source code:

```
0x74,0x6f,0x75,0x63,0x68,0x20,0x2f,0x74,0x6d,0x70,0x2f,0x59,0x4f,0x59,0x4f,
0x59,0x4f,0x0};
```

The above code yields the ASCII characters… touch /tmp/YOYOYO

Signature 4: On all versions of the exploit, the "unpatched" version of the exploit would execute the "/adm/sh –c" command. The following came from character array definitions portion of the source code:

```
0x2f,0x61,0x64,0x6d,0x2f,0x6b,0x73,0x68,0x0,0x2d,0x63
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Conversely, the patch as prescribed by E-Mind, would change this code such that "/bin/sh –c" would be executed in the stack instead. Horizon himself provides a clue to this in his comments.

Signature 5: In all versions of the exploit, the ASCII characters "ADMRocks" is visible. The following line came from the character array definitions portion of source code:

```
0x41,0x44,0x4d,0x52,0x4f,0x43,0x4b,0x53
```

Signature 6: The following came from the "char linuxcode[ ]=" and "char bsdcode[ ]=" portion of the source code:

```
0x70,0x6c,0x61,0x67,0x75,0x65,0x7a,0x5b,0x41,0x44,0x4d,0x5d
```

The above code yields the ASCII characters… plaguez[ADM] .[5]

Signature 7: The following came from the "char linuxcode[ ]=" portion of the ADM-NXT version by Horizon and Plaguez:

```
0x0,0x0,0x0,0x10,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x74,0x68,0x69,0x73,0x69,0x73,
0x73,0x6f,0x6d,0x65,0x74,0x65,0x6d,0x70,0x73,0x70,0x61,0x63,0x65,0x66,0x6f,
0x72,0x74,0x68,0x65,0x73,0x6f,0x63,0x6b,0x69,0x6e,0x61,0x64,0x64,0x72,0x69,
0x6e,0x79,0x65,0x61,0x68,0x79,0x65,0x61,0x68,0x69,0x6b,0x6e,0x6f,0x77,0x74,
0x68,0x69,0x73,0x69,0x73,0x6c,0x61,0x6d,0x65,0x62,0x75,0x74,0x61,0x6e,0x79,
0x77,0x61,0x79,0x77,0x68,0x6f,0x63,0x61,0x72,0x65,0x73,0x68,0x6f,0x72,0x69,
0x7a,0x6f,0x6e,0x67,0x6f,0x74,0x69,0x74,0x77,0x6f,0x72,0x6b,0x69,0x6e,0x67,
0x73,0x6f,0x61,0x6c,0x6c,0x69,0x73,0x63,0x6f,0x6f,0x6c,0xeb,0x86,0x5e,0x56,
```

Lance Spitnzer's forensics was able to obtain the following readible ASCII code"

```
00 00 00 10 00 00 00 00 00 00 00 74 68 69 73 69  ...........thisi
73 73 6F 6D 65 74 65 6D 70 73 70 61 63 65 66 6F  ssometempspacefo
72 74 68 65 73 6F 63 6B 69 6E 61 64 64 72 69 6E  rthesockinaddrin
79 65 61 68 79 65 61 68 69 6B 6E 6F 77 74 68 69  yeahyeahiknowthi
73 69 73 6C 61 6D 65 62 75 74 61 6E 79 77 61 79  sislamebutanyway
77 68 6F 63 61 72 65 73 68 6F 72 69 7A 6F 6E 67  whocareshorizong
6F 74 69 74 77 6F 72 6B 69 6E 67 73 6F 61 6C 6C  otitworkingsoall
69 73 63 6F 6F 6C EB 86 5E 56 8D 46 08 50 8B 46  iscool..^V.F.P.F
```

7. *How to Protect Against the Attack*

- Upgrading to BIND version 8.2.2 patch level 5, or higher, is strongly recommended for all users of BIND. With regard to the subject exploit, this is the easiest and best way to mitigate this attack.
- Change UID and GID of in.named daemon to a non-root UID and GID. This is analogous to why web server run as "nobody".
- A more holistic approach to counter buffer overflows in general, is to practice secure coding practices that employ argument validation routines and "safe" compilers. Also the use of secure routines such as fget( ), strncpy( ), and

[5] A White Paper, authored by Lance Spitzner, Know Your Enemy: A Forensics Analysis , focuses on how SNORT was used as a forensics tool to piece together the actions of a real intruder. This paper greatly facilitated the analysis of the ADM-NXT exploit with regard to signatures 6 and 7.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

strncat( ) will reduce the likelihood of buffer overflows. [6] Security representation on configuration control boards is also necessary and should be a matter of routine whenever any code is modified.

8. Source Code/Pseudo Code

Source code for both the ADM and TESO versions of the BIND 8.2 NXT remote buffer overflow attack can be found in paragraph 3 above.

Pseudo code for this exploit is as follows:

1. Determine if target name serve is vulnerable to NXT exploit via *dig* or *nslookup*
2. Perform DNS queries of target name server in order to determine if target name server performs recursive queries
3. Create subdomain delegation records on name server that is an accomplice to the attack and reinitialize in.named daemon…kill –HUP <in.named pid>
4. Edit source code to change /adm/sh to /bin/sh (in hex) by searching the source code for 0x2f,0x61,0x64,0x6d,0x2f and replacing it with x2f,0x62,0x69,0x6e,0x2f on hacker_host
5. Compile the *t666.c* source code with C compiler on hacker_host
6. Execute the compiled and linked executable on hacker_host
7. Request target name server to perform recursive query in order to resolve a hostname with subdomain that was delegated to hacker_host.
8. hacker_host sends a large NXT record containing code that exploits the remote BIND server memory stack with a buffer overflow.
9. hacker_host gains shell access with privileges as in.named daemon on target name server.
10. Attacker sets up user account and back channel on name server; exits shell.

[6] *Hacking Exposed*, by Stuart McClure, Joel Scambray, and George Kurtz, pp 215-216, Osborne/McGraw Hill, 1999.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Last Updated: April 24th, 2017

Upcoming Training

Click Here to
{Get CERTIFIED!}

Automotive Cybersecurity Summit	Detroit, MI	May 01, 2017 - May 08, 2017	Live Event
SANS Riyadh 2017	Riyadh, Saudi Arabia	May 06, 2017 - May 11, 2017	Live Event
SANS Security West 2017	San Diego, CA	May 09, 2017 - May 18, 2017	Live Event
Community SANS St Louis SEC504	St Louis, MO	May 15, 2017 - May 20, 2017	Community SANS
SANS Northern Virginia - Reston 2017	Reston, VA	May 21, 2017 - May 26, 2017	Live Event
SANS London May 2017	London, United Kingdom	May 22, 2017 - May 27, 2017	Live Event
SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
Community SANS Huntsville - SEC504	Huntsville, AL	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Thailand 2017	Bangkok, Thailand	Jun 12, 2017 - Jun 30, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Philippines 2017	Manila, Philippines	Jun 19, 2017 - Jun 24, 2017	Live Event
Community SANS Marina del Rey SEC504	Marina del Rey, CA	Jun 19, 2017 - Jun 24, 2017	Community SANS
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Community SANS
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
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 Cyber Defence Singapore 2017	Singapore, Singapore	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
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Sacramento SEC504	Sacramento, CA	Jul 17, 2017 - Jul 22, 2017	Community SANS