Use offense to inform defense.
Find flaws before the bad guys do.

Polymorphic, multi-lingual websites: A theoretical approach for improved website security

GIAC (GWAPT) Gold Certification

Author: Jonathan Risto, jonathan.risto@hotmail.com

Advisor: Chris Walker

Accepted: 7/22/2016

Abstract

Web traffic is one of the largest single types of traffic on the internet. From providing information to potential customers to hosting pictures for people to view, a company website is often the first place that people go to get information on that business. With the increased functionality, also comes increased attacks on the corporate websites. From Cross Site Scripting and SQL injection to brute force password attacks, websites need to be protected and use secure code. The problem is that each company needs to hire expert coders to secure their websites, either of which is not always done properly. This paper proposes a new approach to developing websites that are both safer and offer greater protection from the threats through standardized, secure code and by shifting code to use for each page load. Through analysis, it is shown that new approach should theoretically increase the security of websites over what is in use currently.

1. Introduction

With the increased functionality available online that includes blogging, online purchasing, and querying services, websites provide an enormous amount of functionality and integration with numerous servers and computers within the back-end network. However, with the continual increase in functionality that websites offer, comes an augmented attack surface for adversaries to leverage.

Within their 2016 Security Report, Cisco Systems detailed the growth in attackers using compromised websites as their launching point to gain access to corporate networks (Cisco Systems, 2016). Akamai revealed the over 25% increase in total web application attacks in their Q1 2016 State of the Internet report (Akamai, 2016). Moreover, it is not only over HTTP. In fact, for Q1, Akamai saw an increase of over 200% in attacks that were over HTTPS (Akamai, 2016). In the small business landscape, it was noted that access is possible to corporate data due to a lack of policies and poorly coded applications (Baker, 2013).

Companies today develop their web applications independently with developers who may not have significant knowledge of security. In their 2015 Data Breach Report, Verizon provided information that over 9% of confirmed data breaches stemmed from web app attacks (Verizon, 2015).

To help deal with the continued importance of web applications, and the increased exposure that organizations have from this publically facing devices, a new approach to how this information is presented back to the end user is needed. A solid understanding of the current mechanisms used in industry is also required.

jonathan risto;jonathan.risto@hotmail.com

2. Website Data Flows

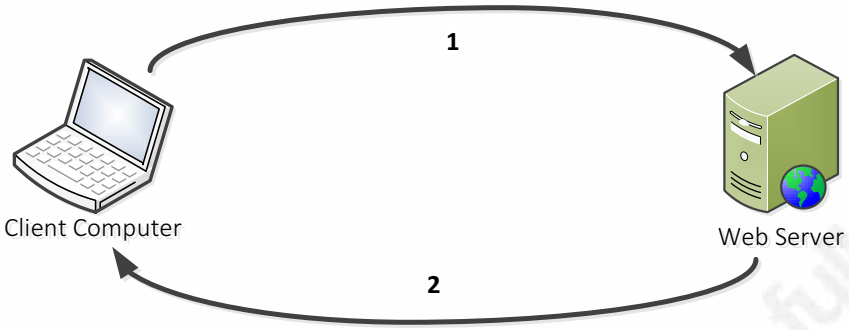

Figure 1 - Basic website data flows

When a user browses to a website, there is an exchange of information. The user sends the request to the server, as in Figure 1, the arrow labeled 1 carries the user requests across to the server. The server performs some processing, and sends the information back to the user, as shown by arrow 2, to be parsed by the user's web browser and finally displayed to the end user. This sequence of events is performed for each piece of information that the user request. Ths is a relatively straightforward and simple process, but it does not allow much beyond static information displayed on the website. The user has no input into the information presented beyond the request sent to the server for a particular page, such as http://www.google.ca. For the attacker, there are also minimal attack vectors to this server, beyond attempting to the break the server by sending requests to the device and seeing what is returned.

For more complex situations, the initial request to the web server is the same, but what happens moving forward changes. Most web pages offer a dynamic display of information, permitting the user to interact with the website in some form. Be it from queries to the site, such as searching for an item, to web commerce, users request certain pieces of information, the companies servers process this and return to the user the dynamic content.

jonathan risto;jonathan.risto@hotmail.com

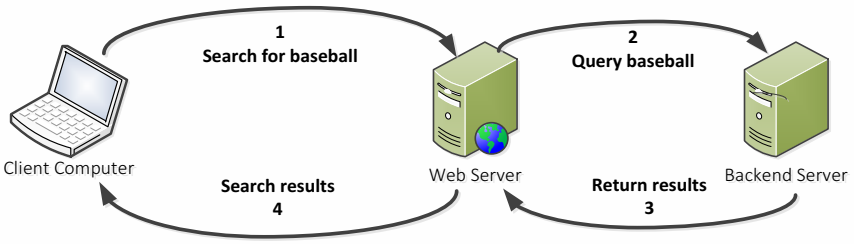

Figure 2 - Complex website data flows

In Figure 2, the user has already loaded the original site, as per Figure 1. However, on this site, the user can request the current prices of any item that they want. In this case, the user has entered baseball as the term to find out what the current price is on baseballs. The user clicks on send (step 1), and the information is forwarded to the web server. The information referenced is then passed along to the back-end server (step 2), that processes the information and returns all the baseball references it found during the search and their associated prices (37 instances in our example case) in step 3. In step 4 the web server packaged the information as needed and sent it to the users browser. The user browser displays the 37 items to the end user. Typically a server uses one language for the front-end communications (such as HTML or CSS) and one language for the back-end communications (such as Python or C).

There are numerous additional vectors that an attacker could use to attempt to break into this network through the web interfaces than in Figure 1. Beyond the attack types of network 1, there are opportunities to break the web server directly through the data input methods, using, for example, a typical buffer overflow method. With information passing to the back-end server, attacks could include data injection attacks, error recovery both against the end database as well as the chosen programming language used to handle these queries.

jonathan risto;jonathan.risto@hotmail.com

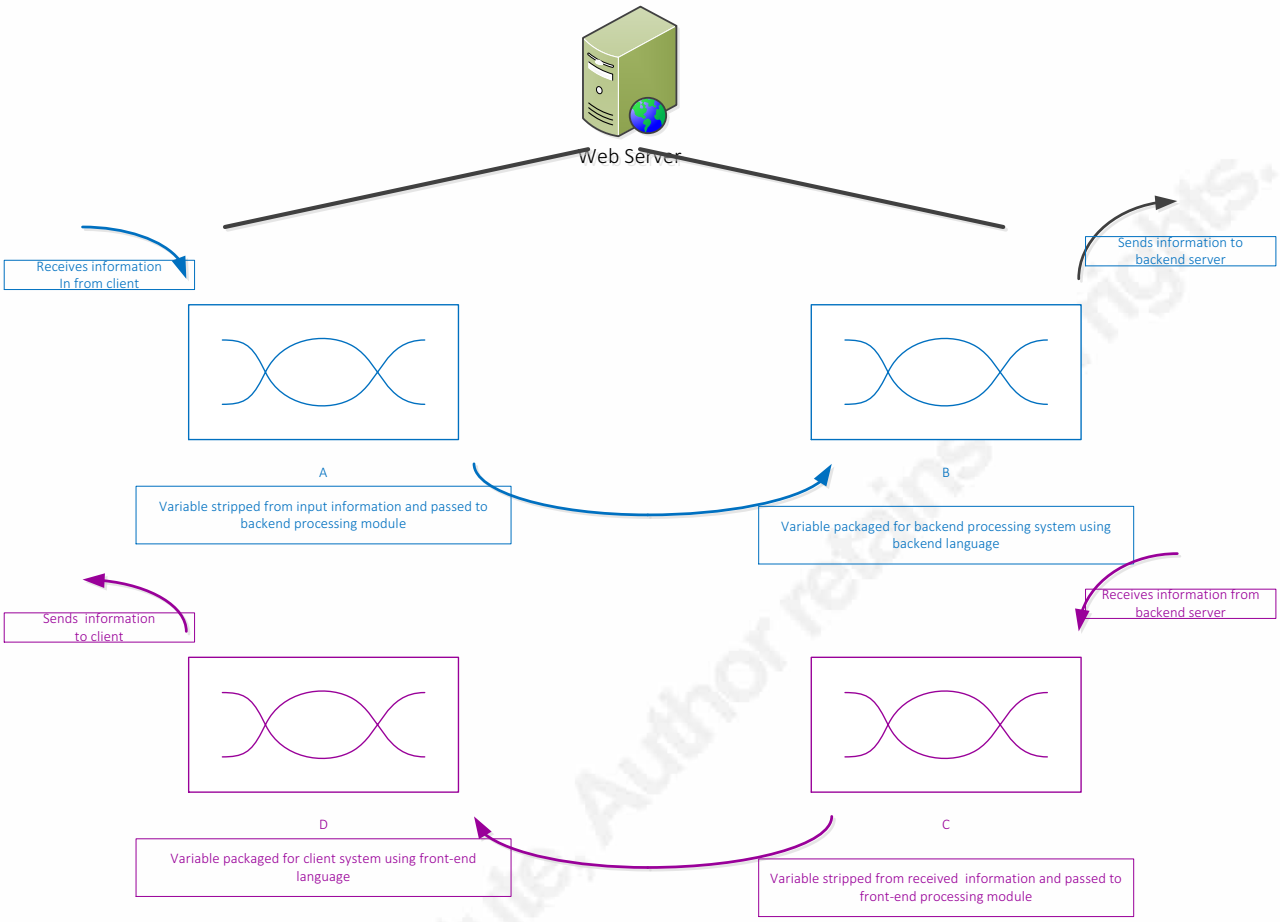

Figure 3 - web server functionality diagram

Focusing on the web server, as shown in Figure 2, there is much functionality going on inside the web server, to handle the web traffic. Expanding it out, and showing some of the relevant functions, is done in Figure 3. The information flow is shown in the top portion of the diagram highlighted in blue, displays the appropriate data manipulations, labeled A and B, which occur between steps 1 and 2. Similarly, the bottom data flow, shown in purple, highlights the relevant data manipulation, marked C and D, between steps 3 and 4.

jonathan risto;jonathan.risto@hotmail.com

2.1. Programming languages used in web development

Numerous front-end languages are used by developers to deliver content to web clients. For this paper, we focus on five popular languages used currently (Monk, 2014). These languages are:

- HTML
- Javascript
- CSS
- Actionscript
- Silverlight

Other languages could be utilized, such as Java applets, or Visual Basic (VB) Scripts, but for this work it has been restricted to 5.

Similar to the client side, there are numerous back-end web languages used to connect the server to other devices in the network. For this work, it has been restricted to the following eight popular (Monk, 2014) options:

- Java
- Python
- PHP
- .NET (using C# or VB)
- Ruby
- Perl
- Javascript
- C/C++

3. Proposed Approach

To make the system harder for the attacker to understand and eventually break into, the number of attempts that the attacker needs to generate to find the problem increases successfully. Let's say that it takes, on average, 1000 times for the attacker to

jonathan risto;jonathan.risto@hotmail.com

be successful in the search for a problem on the website. What needs to happen is that these 1000 attempts need to be increased through some means. If the problem is 4 times harder to solve for the attacker, then it would require 4000 attempts on average to find the problem. This new system would be four times harder than the original web page in question to attack, or inversely, the attacker would have a 0.025% chance of succeeding.

3.1. Randomly selecting a front-end language

To increase the complexity of the web page in question, we need to make the attacker use more attacks to find the same information. If we were able somehow randomly to choose one of the five front-end languages to deliver the content to web browser each time the web page was loaded, it would be a five times increase in complexity from the attackers side.

For example, let's simplify it to just having two different front-end languages for this example. If the first time the website loads it was in HTML and the second time it was in JavaScript, the attacker cannot compare between the two options, as they are different presentation methods and different code attack vectors. In other words, for each type of attack that the attacker wants to try, he would need to attack HTML and JavaScript to determine if the website is vulnerable. If it takes, on average, 1000 attempts before success, this change would require, on average, 2000 attempts before success.

With the number of languages in use for front-end communication, each attack would need to be performed five times to determine if the website if vulnerable to a particular attack. Each of the 1000 attempts would need to be performed five times, which leads to a requirement, on average, of 5000 attempts before the realization of the attacker’s goal.

The following table shows the increased complexity of the system by increasing the number of front-end languages.

jonathan risto;jonathan.risto@hotmail.com

# of languages	Success Rates
1	0.1000%
2	0.0500%
3	0.0333%
4	0.0250%
5	0.0200%

Table 1 – Success rates for front-end language attacks

3.2. Randomly selecting a back-end language

Similar to the front-end language choices, we have eight back-end language options. If each of the languages was randomly chosen to communicate to the back-end servers, we have an eight-fold increase in the complexity of the website. By permitting the site to communicate in this fashion, we have taken the success rate from 0.1% to 0.0125%.

For the proposed system, the following table shows the decrease in attack success as the number of languages increases.

# of languages	Success Rates
1	0.1000%
2	0.0500%
3	0.0333%
4	0.0250%
5	0.0200%
6	0.0167%
7	0.0143%
8	0.0125%

Table 2 –Success rates for back-end language attacks

3.3. Web page segments for added complexity

As discussed in the front-end and back-end language sections, if we randomly choose a language each time the web page is loaded, we increase the complexity for the

jonathan risto;jonathan.risto@hotmail.com

attacker based on the number of languages in use. An alternative method is required to enhance the complexity of any particular web page.

Initially, a web page is segmented into four parts. Moreover, each part loads the content through a randomly selected language; we have four pieces that are independently complex in a similar fashion to the previous sections. However, if we combine these two, the complexity increases based on the number of languages as well as the number of pieces for the web page in question.

For example, if we have four segments on the page with four languages possible for each segment. The complexity of this site is not just equal to the four languages in question as we previously saw. We have segment 1, which has four languages, segment 2 with four languages, segment 3 with four languages and finally segment 4 with four languages. To find a problem with a particular section, it would require 4000 attempts, similar to what we saw previously. However, if we are looking for across the entire website, we would need to try 4 x 4000 times, or 16,000 attempts to find the problem, a 16 x increase in complexity. Alternatively, the attacks would have a 0.0063% chance of finding a problem, compared to the standard 0.1% chance

Furthering this along, what happens if each section needs to call on a back-end server for functionality, and each of those calls is randomized. In the example above, we have four segments, each with four languages. Now we add that each section can call upon one of 4 back-end languages randomly. We have four segments, each calling 1 of 4 front-end languages, that in turn can call upon 1 of 4 back-end languages. This would equal 4 x 4 x 4 = 64 times the complexity.

3.4. Standardized code for standardized functionality

With the above approaches, it would be too complex for a developer using current technologies and methods to create a web page that would permit the languages to be changed each time loaded. So this would require the development of a new system/software that would provide the following functionality to the community.

First, it needs to be able to segment the page that is sent out to the browser. It also needs to select dynamically the front-end language that is to be used for each of the

segments. It also needs to choose the - language utilized for each segment. Finally, it needs to track all of this for each web page that is displayed. For the system to accomplish this, a fundamental change is required from how web pages are currently coded and designed.

If a web page designer was able to choose from functionality needed, such as user login, and place this information where they would like to have it on the site instead of having to write the Javascript or CSS code themselves, it becomes more of a flow chart design method for the web page. If the server provided the standardized and secure code for the user login in each of the front-end languages possible, then the server would be able to send to the browser the user login in any of the front-end languages. It knows that the designer wished to have a user login, and it randomly chooses between the available options and sends this to the web browser.

Similar for back-end programming languages, whenever there is a call to the back-end systems, the server needs to know what device to call. It also needs to know what information is being passed to it (our example case was sending a query of baseball) and what information returns from the server (we returned 37 results). Again, if the server knows that there is a query dialogue presented to the user, a means to submit, and then a query needs to be passed to the particular back-end server, the server could have different query types pre-programmed and available to be used to circulate the required information. Again, the web page designer does not need to know the code to use, but rather that there is information to pass along and results to be displayed. The first time the server may use PHP for this communication, and the second time it may use Python. The end user does not care how this is passed along, just so long as the query returns the desired information.

By using a common library of routines to perform actions on the web page, as well, to pass information to the back-end servers, several security problems are alleviated or significantly minimized. The majority of security flaws can be removed from the software in question through more rigorous testing and tighter control on the code in question. By doing this, it helps resolve the 20 Critical Controls Application Software Security issue (SANS, 2016) and should assist with eliminating the OWASP 10 Most Critical Web Application Security Risks (OWASP, 2013).

jonathan risto;jonathan.risto@hotmail.com

3.5. Putting it all together

A standard system, such as shown in Figure 2, typically uses one language for front-end communications and one language for back-end communications. For this type of setup, if an attacker needs 1000 attempts before he is successful in breaking into the system, then there is 1/1000 or 0.1% likelihood of success.

Moving to a system like the one proposed; let's assume there are four front-end languages in use, meaning that our attacker would need 4 * 1000 attempts to be successful, or 1/4000 times, equating to a 0.025% likelihood of success. Alternatively, 4 times less likely to succeed.

Now, the back-end system communications need to be examined. We have four back-end languages. Let's say that for the attack to work, it needs 1 of the back-end languages to be used; the success rate would be 1/4000 again for this options (4 languages x 1000 attempts).

Now if we put this all together, the complete end-to-end situation assumes that you need a particular front-end language and an explicit back-end language to be in use for the attack to work. Following this assumption that our attacker needed both the front-end and back-end languages to be in use for his success and if we have four front-end and four back-end languages in use, we have numerous combinations of front-end and back-end languages possible. For each front-end language, we have four back-end languages. So there are 16 combinations in total for how the languages would pair up, meaning that the attacker has a 1/16000 chance, or a 0.00625% chance of success. In other words, this attack is 16 times less likely to success than today's situation.

Using the five standard front-end languages and the eight common back-end languages, this would lead us to a website that would require, on average, 40 times more work to break into than current sites, assuming that there is still a 0.1% chance of success for the attacks.

Placing segmentation on the page, with four segments on the web page in question, we have five front-end languages, eight back-end languages, and four

jonathan risto;jonathan.risto@hotmail.com

segmentations that would require, on average, 160 times more work to break into than current sites.

With the proposed system, with a consistent, more robustly checked code base, the possibility of a successful attack working should diminish as well. If it is assumed that the system would be ten times harder to hack, based on the effort used to create the language translator, this creates a system that is 1600 times more secure than current websites.

	number of attacks needed
standard page	1,000
5 front languages	5,000
8 back languages	8,000
4 page segments	4,000
combined	160,000

Table 3 - Attacker attempts required

Another way to look at this would be in time duration. If an attacker wishes to evade detection and does one query every 10 minutes, or in other words, six attack attempts an hour, or 144 a day, it would require just under seven days for the offensive to be successful with 1000 attack attempts.

Using the same rate of attack, and a site that is 1600 times more secure, it would require over 11,111 days. Even if the attacker increases their rate of attack to once a minute, it would still take 1,111 days. To reduce the timeframe for a successful attack in the same seven days timeframe, an attacker would need to be sending, on average, 2.6 queries each and every second to the web server in question for the entire seven days to be successful.

3.6. Threat intelligence integration

An additional means to continue to thwart the attacks against websites with this approach would be to factor in threat intelligence, be it open or closed source, as a

jonathan risto;jonathan.risto@hotmail.com

weighting factor. If a means to factor in new vulnerabilities within a specific language or application was possible, it would further enable the system to adapt dynamically change to the current threat landscape. If the system were able, without human intervention, to modify the selected language based on current threat vectors being used, it would significantly augment the value brought by the system.

For example, if there was a new vulnerability discovered in Java, the system could factor in this new information to reduce the weighting factor on selecting Java from a $1/8^{th}$ or 12.5% to a lower percentage, to only a 6.25% or by whatever factor is decided as appropriate. By automatically reducing the rate at which the language is presented, the known problems are not offered to an attacker and lowers the likelihood of exploitation.

Another means through which threat intelligence could be utilized would be to factor in current attack trends being seen. Often an increase in certain traffic types indicates a newly published vulnerability or a possible unpublished vulnerability that is being widely exploited in networks. By having the ability to factor in likely threat vectors, the system again could lower the likelihood that javascript is used, helping protect the system from the newest threat vector.

4. Benefits and Drawbacks Analysis

The described method provides some advantages if implemented and used for web page loading and back-end processing of web queries. Firstly, the ability to substantially increase the security of the system in question is challenging to overlook. If realizable, this would provide improved security to any site that utilizes it. Even if the estimated numbers are only one quarter attainable, an increase in the security by 400 times cannot be ignored.

Another benefit of the proposed approach is that it simplifies the web page creation process, enabling the majority of individuals to be able to create a web page from this ‘block diagram’ approach, without having to worry about the protection of their creation. The security of the solution is provided outside of the design work.

jonathan risto;jonathan.risto@hotmail.com

Regardless of the benefits of the solution, there are drawbacks as well. First and foremost, it requires new web server software to implement this solution. As with any project of this size, it is not an insignificant task to create this software.

Another factor in considering this solution is the inherent problem of the use of standard code in a significant number of locations. As the industry experienced in 2014, millions of websites had a security issue when the OpenSSL software bugs were discovered and publicized (Ashford, 2014). A thorough analysis is needed to ensure that potential impacts are understood.

A final concern relates to the scalability of the solution. While this design may function on a small scale, analysis of an implementation is needed to determine if difficulties exist scaling this from a few requests to thousands of requests.

5. Way Forward/Conclusion

Within this paper, a new and innovated approach to web page security and design has been presented. While this method does require that new software is developed to accommodate the work, the increase in security would outweigh the potential changes needed for an enterprise environment. The additional obfuscation and dynamic loading of the web pages should require no changes to client-side software; this does need to be proven through demonstration and experimentation.

Numerous steps must be completed to move this idea forward. One of the first items necessary would be a prototype system and software that can perform the functions outlined. This prototype would provide the proof of concept necessary to test the feasibility of the solution in question. A prototype would also provide a test environment to determine how involved the coding is, and if the web page design process would be simplified.

As mentioned within the paper, there is a possibility that scalability could be a concern with the solution for websites that a large volume of transactions each second. A prototype assists with this investigation, as would analysis of the system to determine processing bottlenecks and areas where further optimization is required.

jonathan risto;jonathan.risto@hotmail.com

jonathan risto;jonathan.risto@hotmail.com

References

Akamai. (2016). *State of the Internet security Q1 2016*. Retrieved July 12, 2016, from Akamai: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-q1-2016-state-of-the-internet-security-report.pdf

Ashford, W. (2014, April 8). *OpenSSL security flaw could affect millions of websites, warn researchers*. Retrieved July 7, 2016, from Computer Weekly: http://www.computerweekly.com/news/2240217758/OpenSSL-security-flaw-could-affect-millions-of-websites-warn-researchers

Baker, C. (2013, September 23). *10TH NATIONAL SMALL BUSINESS CONFERENCE.* Retrieved July 12, 2016, from National Defence Industrial Association: http://dtic.mil/ndia/2013smallbusiness/Baker.pdf

Cisco Systems. (2016). *Cisco 2016 Annual Security Report*. Retrieved July 12, 2016, from Cisco Systems: http://www.cisco.com/c/dam/assets/offers/pdfs/cisco-asr-2016.pdf

Monk, P. (2014, September). *Which programming languages are front-end and which ones are back-end?* Retrieved July 5, 2016, from Quora: https://www.quora.com/Which-programming-languages-are-front-end-and-which-ones-are-back-end

OWASP. (2013). *OWASP Top 10.* Retrieved July 5, 2016, from OWASP: file:///C:/Users/Jonathan/Downloads/OWASP%20Top%2010%20-%202013.pdf

SANS. (2016). *20 Critical Controls - 18 Application Security*. Retrieved July 5, 2016, from SANS Institute: https://www.sans.org/critical-security-controls/vendor-solutions/control/18

Verizon. (2015, April 26). *2015 Data Breach Investigations Report.* Retrieved July 5, 2016, from Verizon: http://www.verizonenterprise.com/DBIR/

jonathan risto;jonathan.risto@hotmail.com

Last Updated: March 25th, 2019

Upcoming SANS Penetration Testing

SANS 2019 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Orlando, FL	Apr 01, 2019 - Apr 06, 2019	vLive
SANS 2019	Orlando, FL	Apr 01, 2019 - Apr 08, 2019	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	SEC504 - 201904,	Apr 02, 2019 - May 09, 2019	vLive
SANS London April 2019	London, United Kingdom	Apr 08, 2019 - Apr 13, 2019	Live Event
SANS Riyadh April 2019	Riyadh, Kingdom Of Saudi Arabia	Apr 13, 2019 - Apr 18, 2019	Live Event
SANS Seattle Spring 2019	Seattle, WA	Apr 14, 2019 - Apr 19, 2019	Live Event
SANS Boston Spring 2019	Boston, MA	Apr 14, 2019 - Apr 19, 2019	Live Event
Mentor Session - SEC560	Columbia, MD	Apr 18, 2019 - Jun 20, 2019	Mentor
Community SANS Phoenix SEC504	Phoenix, AZ	Apr 22, 2019 - Apr 27, 2019	Community SANS
Northern Virginia- Alexandria 2019 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Alexandria, VA	Apr 23, 2019 - Apr 28, 2019	vLive
SANS Northern Virginia- Alexandria 2019	Alexandria, VA	Apr 23, 2019 - Apr 28, 2019	Live Event
Mentor Session - SEC504	Baltimore, MD	Apr 25, 2019 - Jun 13, 2019	Mentor
SANS Muscat April 2019	Muscat, Oman	Apr 27, 2019 - May 02, 2019	Live Event
Cloud Security Summit & Training 2019	San Jose, CA	Apr 29, 2019 - May 06, 2019	Live Event
SANS Pen Test Austin 2019	Austin, TX	Apr 29, 2019 - May 04, 2019	Live Event
Community SANS Minneapolis SEC504	Minneapolis, MN	Apr 29, 2019 - May 04, 2019	Community SANS
SANS vLive - SEC560: Network Penetration Testing and Ethical Hacking	SEC560 - 201904,	Apr 30, 2019 - Jun 06, 2019	vLive
Mentor Session @Work - SEC504	Melbourne, FL	May 01, 2019 - May 10, 2019	Mentor
Mentor Session - SEC560	Chantilly, VA	May 01, 2019 - Jun 26, 2019	Mentor
Mentor @Work - SEC504	Mechanicsburg, PA	May 02, 2019 - Jun 20, 2019	Mentor
Community SANS Ottawa SEC504	Ottawa, ON	May 06, 2019 - May 11, 2019	Community SANS
SANS Bucharest May 2019	Bucharest, Romania	May 06, 2019 - May 11, 2019	Live Event
Community SANS Denver SEC504	Denver, CO	May 06, 2019 - May 11, 2019	Community SANS
Security West 2019 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	San Diego, CA	May 09, 2019 - May 14, 2019	vLive
Mentor Session - SEC560	Boca Raton, FL	May 09, 2019 - May 24, 2019	Mentor
Security West 2019 - SEC560: Network Penetration Testing and Ethical Hacking	San Diego, CA	May 09, 2019 - May 14, 2019	vLive
SANS Security West 2019	San Diego, CA	May 09, 2019 - May 16, 2019	Live Event
Security West 2019 - SEC542: Web App Penetration Testing and Ethical Hacking	San Diego, CA	May 09, 2019 - May 14, 2019	vLive
SANS Stockholm May 2019	Stockholm, Sweden	May 13, 2019 - May 18, 2019	Live Event
SANS Perth 2019	Perth, Australia	May 13, 2019 - May 18, 2019	Live Event
Community SANS Falls Church SEC560	Falls Church, VA	May 13, 2019 - May 18, 2019	Community SANS