Interested in learning more about security?

SANS Institute InfoSec Reading Room

An Overview of Session Hijacking at the Network and Application Levels

With the business of ecommerce booming, more and more sensitive information is being passed around on the web. Financial and identity information are constantly at risk of being stolen as more and more users take advantage of the ease of doing business online through web applications. The purpose of this paper is to discuss one particularly salient security threat that this creates: session hijacking. It is important to understand this threat and to make an effort to design networks and applications that will be less v...

An Overview of Session Hijacking at the Network and Application Levels
By Mark Lin
Date Submitted: 1/18/2005
GSEC Practical Assignment v1.4c (Option 1)

Abstract

With the business of ecommerce booming, more and more sensitive information is being passed around on the web. Financial and identity information are constantly at risk of being stolen as more and more users take advantage of the ease of doing business online through web applications. The purpose of this paper is to discuss one particularly salient security threat that this creates: session hijacking. It is important to understand this threat and to make an effort to design networks and applications that will be less vulnerable to it. Sensitive user information is stored within each session that is created upon client authentication and hackers are willing to go to great lengths to steal it. "Indeed, in a study of 45 Web applications in production at client companies, @Stake (recently acquired by Symantec) found that 31 percent of e-commerce applications were vulnerable to cookie manipulation and session hijacking." [9] In this paper, I will be setting the stages for the session hijacking to occur, then discussing the techniques and mechanics of the act of session hijacking, and finally providing general strategies for its prevention.

The Stages

Before we can discuss the intricacies of session hijacking, we need to be familiar with the stages on which this act plays out. We have to identify the vulnerable protocols and also obtain an understanding of what sessions are and how they are used.

Based on my research, I have found that the three main protocols that manage the data flow on which session hijacking occurs are TCP, UDP, and HTTP, though other protocols that do not use encryption (e.g. telnet, FTP, DNS) also can be vulnerable.

TCP is an abbreviation for Transmission Control Protocol.

> Webopedia defines it as "one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent." [17]

The last part of Webopedia's definition is important in our discussion of session hijacking. In order to guarantee that packets are delivered in the right order, TCP uses acknowledgement (ACK) packets and sequence numbers to create a "full duplex reliable stream connection between two end points," [3] with the end points referring to the communicating hosts. The two figures

below provide a brief description of how TCP works:

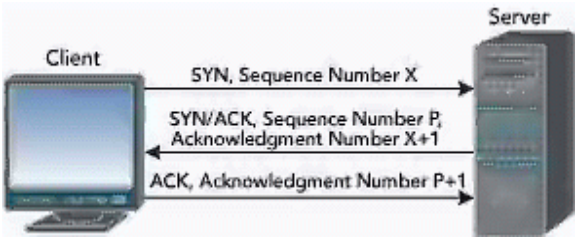

Figure 1 TCP Three-Way Handshake
(Figure and TCP summary taken from Lam, LeBlanc, and Smith) [8]

The connection between client and server begins with a three-way handshake (Figure 1). It proceeds as follows:

- Client sends a synchronization (SYN) packet to the server with initial sequence number X.
- Server responds by sending a SYN/ACK packet that contains the server's own sequence number p and an ACK number for the client's original SYN packet. This ACK number indicates the next sequence number the server expects from the client.
- Client acknowledges receipt of the SYN/ACK packet by sending back to the server an ACK packet with the next sequence number it expects from the server, which in this case is p+1.

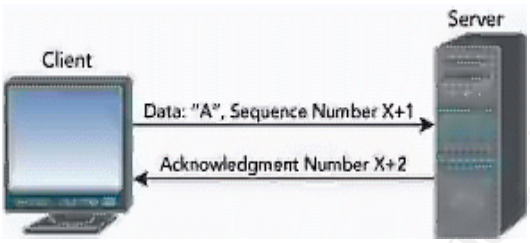

Figure 2 Sending Data over TCP
(Figure and TCP summary taken from Lam, LeBlanc, and Smith) [8]

After the handshake, it's just a matter of sending packets and incrementing the sequence number to verify that the packets are getting sent and received. In Figure 2, the client sends one byte of info (the letter "A") with the sequence number X+1 and the server acknowledges the packet by sending an ACK packet with number x+2 (x+1, plus 1 byte for the A character) as the next sequence number expected by the server. The period where all this data is being sent over TCP between client and server is called the TCP session. It is our first stage on which session hijacking will play out.

The next protocol is UDP, which is an abbreviation for User Datagram Protocol.

> Webopedia defines it as "a connectionless protocol that, like TCP, runs on top of IP networks. Unlike TCP/IP, UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagrams over an IP network." [17]

UDP doesn't use sequence numbers like TCP. It is mainly used for

broadcasting messages across the network or for doing DNS queries. Online first person shooters like Quake and Half-life make use of this protocol. Since it's connectionless and does not have any of the more complex mechanisms that TCP has, it is even more vulnerable to session hijacking. The period where the data is being sent over UDP between client and server is called the UDP session. UDP is our second stage for session hijacking.

HTTP stands for Hyper Text Transfer Protocol.

> Going back to Webopedia, we define HTTP as "the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. " [17]

It is also important to note that HTTP is a stateless protocol. Each transaction in this protocol is executed independently with no knowledge of past transactions. The result is that HTTP has no way of distinguishing one user from the next. To uniquely track a user of a web application and to persist his/her data within the HTTP session, the web application defines its own session to hold this data. HTTP is the final stage on which session hijacking occurs, but unlike TCP and UDP, the session to hijack has more to do with the web application's implementation instead of the protocol (HTTP).

The Victims

The stages are set, so we define the victims next. The victims are the sessions that get hijacked.

For TCP and UDP, the sessions are the periods of time where the client is connected and actively passing information to the server. At the beginning of the session, the user/client is authenticated and then it is assumed that as long as the ACK numbers on the packets are correct, the server is talking to the same user. To distinguish this from the HTTP session, some author's label this the "TCP/IP stream." [16] The TCP and UDP sessions are hijacked by taking control of the packets that are sent between client and server.

For HTTP, the sessions are the periods of time where the user is accessing a web application, from user logon to user logoff. The HTTP sessions are related to and distinguished from TCP sessions in that "requests from a single user can come over different TCP/IP streams, directly or through proxies, or even from different IP addresses." [16] The session is all the interaction the user has with the application, regardless of the TCP/IP streams its data travels on. As I have stated earlier, HTTP is a stateless protocol. The result is that web application sessions have to be kept track of separately from the protocol. How this is implemented is dependent on the web application. In general, "when a user logs into an application a session is created on the server to maintain the state for other requests originating

from the same user." [5] The session stores all necessary parameters and identity information for the particular user it's associated with. It is kept in memory and cached until the user logs out of the application or is inactive for a predefined period of time.

It is important that we understand how sessions are commonly managed at this level. To identify a session, the session is given a session ID. This ID is sent between client and server for those HTTP requests that belong to that session. The web application usually implements session management as either client-side or server-side. In the client-side session implementation, the bulk of authorization and identity information for the user is stored client-side in a cookie. The client sends the information found in its cookie (including the session ID) to let the server know who is sending these requests. In contrast, the server-side implementation stores the bulk of authorization and identity information in a back-end database on the server. The session ID is used to index the user's information in that database, so the server can access the appropriate information upon receiving requests from the client.

The Act: Session Hijacking Defined

We have now defined the 3 stages and the victims on these stages. The next step is to define the act of session hijacking within the context of these stages.

Simply put, session hijacking is "a security attack on a user session over a protected network."[17] It involves employing various techniques to tamper with or take over TCP and web application user sessions. If the session hijacker successfully impersonates the user/client, he gains access to the sensitive information found in the session.

The Levels

Session hijacking occurs on two levels: the network level and application level. Of the three stages we've defined already, we can classify the TCP and UDP stages under the network level and the HTTP stage under the application level. In terms of our discussion, the levels represent the two different types of sessions that can be hijacked. The network level refers to the interception and tampering of packets transmitted between client and server during a TCP or UDP session. The application level refers to obtaining session IDs to gain control of the HTTP user session as defined by the web application. Attacks at each level are not unrelated, however. Most of the time, they will occur together depending on the system that is attacked. For example, a successful attack on as TCP session will no doubt allow one to obtain the necessary information to make a direct attack on the user session on the application level.

Network Level

Network level session hijacking is particularly attractive to hackers, because they do not have to customize their attacks on a per web application basis. It is an attack on the data flow of the protocol, which is shared by all web applications.

TCP Hijacking

The goal of the TCP session hijacker is to create a state where the client and server are unable to exchange data, so that he can forge acceptable packets for both ends, which mimic the real packets. Thus, he is able to gain control of the session. At this point, the reason why the client and server will drop packets sent between them is because the server's sequence number no longer matches the client's ACK number and likewise, the client's sequence number no longer matches the server's ACK number. In "Simple Active Attack Against TCP," Laurent Joncheray refers to this state as the "desynchronized state" whereas the state where the TCP session is open and data is being exchanged is referred to as the "established state." [7] To achieve the creation of the desynchronized state, the hijacker can employ different techniques:

IP Spoofing: Source Routed Packets

IP spoofing is "a technique used to gain unauthorized access to computers, whereby the intruder sends messages to a computer with an IP address indicating that the message is coming from a trusted host." [17] The trusted host, in the case of session hijacking, is the client. In employing this technique, the session hijacker obtains the IP address of the client and modifies packet headers to indicate that they come from that IP address. This technique allows the hijacker to create his/her own acceptable packets to inject into the TCP Session. The packets are source-routed, meaning that the sender specifies the route the packet will take to get to the destination IP. Using these source-routed packets, the hijacker can route the packets to his host and fool the server into thinking it is communicating with the victim (the client).

Once the hijacker has successfully spoofed an IP address, he determines the next sequence number that the server expects and uses it to inject the forged packet into the TCP session before the client can respond. By doing so, he creates the "desynchronized state." The sequence and ACK numbers are no longer synchronized between client and server, because the server registers having received a new packet that the client never sent. Sending more of these packets will create an even greater discrepancy between the two hosts.

Blind Hijacking

If source routing is disabled, the session hijacker can also employ blind hijacking where he injects his malicious data into intercepted communications in the TCP session. It is called "blind" because the hijacker can send the data or commands, but cannot see the response. The hijacker

is basically guessing the responses of the client and server. An example of a malicious command a blind hijacker can inject is to set a password that can allow him access from another host.

Man in the Middle: Packet Sniffer

This technique involves using a packet sniffer that intercepts the communication between the client and server. With all the data between the hosts flowing through the hijacker's sniffer, he is free to modify the content of the packets. The trick to this technique is to get the packets to be routed through the hijacker's host. In "Theft On The Web: Prevent Session Hijacking," Lam, LeBlanc, and Smith list two "tricks" that hijackers use to redirect traffic to create this "man in the middle" situation. [8]

The first technique is to use forged ICMP (Internet Control Message Protocol) packets to redirect traffic between client and server through the hijacker's host. ICMP is an extension of IP that is used primarily to send error messages indicating problems processing packets through a connection. In this case, the hijacker is forging messages to fool the client and server into thinking that the route through his host is better than the original path (better as in faster, shorter, or non-error prone).

The second technique is ARP spoofing. ARP stands for Address Resolution Protocol. ARP tables are used by each host to map local IP addresses to hardware addresses or MAC addresses. ARP spoofing involves sending out forged ARP replies to fool the host broadcasting the ARP request into updating his ARP table, mapping the IP to be impersonated to the hijacker's hardware address. All traffic intended for that IP will be delivered to his host instead. He can then choose to alter and forward the packets to the real host.

UDP Hijacking

UDP session hijacking works the same as TCP session hijacking, except that since UDP is a weaker protocol that does not use sequence or ACK numbers, the hijacker simply has to forge a server reply to a client UDP request before the server can respond. If the "man in the middle" situation exists, this can be very easy for the hijacker, since he can also stop the server's reply from getting to the client in the first place.

Side Effect: TCP ACK Packet Storms

One of the side effects of TCP session hijacking is the occurrence of a TCP ACK Packet Storm. If the hijacker isn't careful, his actions can be very visible, since a TCP ACK Storm can seriously disrupt a network.

A TCP ACK Packet Storm occurs as a result of the "desynchronized state" of the connection between client and server. At this point, client and server are unable to send each other a packet that can be accepted because their expected sequence numbers are no longer synchronized. In order to get resynchronized, they will continue to send each other ACK packets with

numbers that are unacceptable to the other. The cycle of sending these ACK packets back and forth creates a TCP ACK Packet Storm. This can begin to degrade network performance and an intrusion detection system will register this problem rather quickly.

Covering His Tracks: ARP Table Modification and TCP Re-synchronizing

Due to TCP ACK Storms, the TCP session hijacker also has to employ a few tricks to cover his tracks. To quiet the TCP ACK Storm, for instance, the hijacker can alter the ARP tables. The hijacker can forge ARP replies to fool the hosts into assigning nonexistent hardware addresses to the IP's involved in the session hijacking. Since the packets would no longer have anywhere to go, the storm would no longer exist.

Once the hijacker has completed his attack, he can further cover his tracks by re-synchronizing the two affected hosts. In order for the hosts to be re-synchronized, the client's sequence number needs to match the server's next expected number. One way is to send a forged message requesting that the user send the exact number of extra bytes that will make the sequence numbers match. Lam, LeBlanc, and Smith provide the following example message: [8]

```
msg from root: power failure – try to type 13 chars
```

Unfortunately for the hijacker, this requires the user to comply, so it is generally a weak technique.

Application Level

In the application level, the session hijacker not only tries to hijack existing sessions, but also tries to create new sessions using stolen data. Session hijacking at the application level mainly involves obtaining a valid session ID by some means in order to gain control of an existing session or to create a new unauthorized session.

Obtain Session IDs

Application level session hijacking is all about obtaining the session ID, since web applications key off of this value to determine identity.

Session IDs generally can be found in three locations [10]:

- Embedded in the URL, which is received by the application through HTTP GET requests when the client clicks on links embedded with a page.
- Within the fields of a form and submitted to the application. Typically the session ID information would be embedded within the form as a hidden field and submitted with the HTTP POST command.
- Through the use of cookies.

All three of these locations are within the reach of the session hijacker. Embedded session info in the URL is accessible by looking through the browser history or proxy server or firewall logs. A hijacker can sometimes re-enter in the URL from the browser history and get access to a web application if it was poorly coded. Session info in the form submitted through the POST command is harder to access, but since it is still sent over the network, it can still be accessed if the data is intercepted. Cookies are accessible on the client's local machine and also send and receive data as the client surfs to each page. The session hijacker has a number of ways to guess the session ID or steal it from one of these locations.

Observation (Sniffing)

Using the same techniques as TCP session hijacking, the hijacker can create the "man in the middle" situation and use a packet sniffer. If the HTTP traffic is sent unencrypted, the session hijacker has traffic redirected through his host where he can examine the intercepted data and obtain the session ID. Unencrypted traffic could carry the session ID and even usernames and passwords in plain text, making it very easy for the session hijacker to obtain the information required to steal or create his own unauthorized session.

Brute Force

If the session ID appears to be predictable, the hijacker can also guess the session ID via a brute force technique, which involves trying a number of session IDs based upon the pattern. This can be easily set up as an automated attack, going through multiple possibilities until a session ID works. "In ideal circumstances, an attacker using a domestic DSL line can potentially conduct up to as many as 1000 session ID guesses per second." [10] Therefore, if the algorithm that produces the session ID is not random enough, the session hijacker can obtain a usable session ID rather quickly using this technique.

Misdirected Trust

The last technique is what Gunter Ollman refers to as "misdirected trust." [10] It refers to using HTML injection and cross-site scripting to steal session information. HTML injection involves finding a way to inject malicious HTML code so that the client's browser will execute it and send session data to the hijacker. Cross-site scripting has the same goal, but more specifically exploits a web application's failure to validate user-supplied input before returning it to the client system. "Cross-site" refers to the security restrictions placed on data associated with a web site (e.g. session cookies). The goal of the attack is to trick the browser into executing injected code under the same permissions as the web application domain. By doing so, he can steal session information from the client side. The success of such an attack is largely dependent on the susceptibility of the targeted web application.

Countermeasures

There are a number of things that can be done to prevent session hijacking. In the same manner that the act of session hijacking was discussed, I will discuss the countermeasures for session hijacking in terms of the techniques that apply to the network and application levels.

Network Level

At the network level, it's all about protecting your packets. The main strategy to foil session hijackers is to make your packets harder to interpret. If the hijacker cannot decipher the packet headers, the hijacker will not be able to inject malicious packets.

Encrypted Transfer Protocols

The main countermeasure that will make your packets harder to interpret is to implement encrypted transport protocols such as Internet Protocol Security (IPSec), Secure Socket Layers (SSL), and Secure Shell (SSH).

> "An attacker attempting to hijack a session by tunneling in an encrypted transport protocol must, at a minimum, know the session key used to protect that tunnel, which should be difficult to guess or steal." [8]

Encrypted transfer protocols make the session hijacker's job considerably harder. It adds the extra steps of figuring out how to decrypt the communications and how to encrypt the hijacker's own malicious packets so that they make sense to the hosts. Any packets the hijacker sends without using the session key will be dropped by the hosts as undecipherable.

IPSec includes a set of protocols that were developed to ensure the secure exchange of packets at the IP layer. It has two encryption modes: Transport and Tunnel. Transport mode encrypts only the data portion of each packet, but leaves the packet header untouched. The Tunnel mode encrypts both the header and the data portion. In both modes, IPSec provides security against session hijacking, but the Tunnel mode is probably the most ideal since it encrypts the packet header, making it difficult for the hijacker to even see where the packet is coming from and going to. IPSec is a protocol that protects at the network level, since it's mainly concerned with encrypting packet information.

SSL protects private web documents that are sent on the SSL connection. Most browsers support it. By convention, URLs that require an SSL connection start with https: instead of http:. SSL protects more at the web application level, since it encrypts the data sent over the HTTP session.

SSH is also worth mentioning, because it can protect a network from IP spoofing and IP source routing, which are common techniques used by

session hijackers. “A hijacker who has managed to take over a network can only force SSH to disconnect. He or she cannot play back the traffic or hijack the connection when encryption is enabled.” [17]

Disabling the Hijacker’s Tricks

Besides making the packets harder for the hijacker to decipher, you can also lower the possibility of session hijacking by disabling the hijacker’s bag of tricks. The IP Spoofing and source routed packet techniques can be rendered useless if you set up more secure firewalls and routers. These devices can be set up to fight off such attacks. For example, you could set up your rules to ignore source-routed packets when appropriate or even turn off source-routing completely. ARP Spoofing can be prevented by changing the way you configure your network. You can choose to use static ARP tables that don’t rely on ARP requests or if you still want to use ARP, you can utilize tools like “arpwatch,” which allow you to monitor changes in the ARP table. You could also turn off ICMP redirects to make it harder for the hijacker to redirect traffic through his host to create the man in the middle situation.

Application Level:

While session hijacking prevention at the network level involves securing packets, prevention at the application level involves protecting the session ID from being guessed or stolen.

The Strong Session ID

The key to protecting your web application from session hijackers is to make sure that you have implemented a strong session ID. This applies to both client and server side session management. The following are good techniques to remedy the common failings in the implementation of session IDs in web applications. They are compiled from Gunter Ollman’s paper on “Web Session Management” [10], Webkreator’s “Advanced Session Management” [16], Jeff Prosise’s “Wicked Code: Foiling Session Hijacking Attempts”[12], and Rohyt Belani’s “Basic Web Session Impersonation”[1].

1. *Increase the length of the cookie or session ID* – The session ID should be of sufficient length to make it infeasible that the brute force guessing method will successfully obtain a valid ID. The hijacker should not be able to come close to covering the full range of possible session IDs before the session expires, even if he uses an automated script to conduct the attack. Given current processor and bandwidth limitations, Gunter Ollman recommends at least 50 characters. [10]
2. *Make the session ID more random* – It would feel natural to use a sequential session ID or an index number associated with a user as a session ID to distinguish one user from the next, but it makes it way too easy to guess the a valid session ID. Thus, it is recommended that the session ID be able to pass statistical tests of randomness. It should be very difficult to predict the next value that will be generated by the random algorithm.

3. *Protect the integrity of the Session ID* – Append a "message authenticity code" [1] to the session ID to verify that the session ID has not been tampered with. Verify the authenticity of the session ID on each request on the server-side. By doing this, you make it harder for the hijacker to manipulate cookies or the session ID. Examples of message authenticity codes that can be used include an MD5 sum of the original session token's value as sent out by the server, the IP address of the client, and an encoded time stamp of when the session started.
4. *Use server supplied session IDs* – Instead of having your web application create its own session IDs, you can use application server provided session IDs. Examples of such session ids are ASPSESSIONID and JSESSIONID. These session IDs are generally session IDs that have been proven to be less vulnerable to session hijacking.
5. *Use encrypted session IDs* – You can further protect your session IDs by encrypting them. You can write code that encrypts each session token or encrypt the entire channel using SSL. Ollman actually even suggests using different session IDs for secure vs. insecure parts of the site. [10] So, the session IDs that travel on the SSL protected channels would be different from the ones that travel on the unprotected channels.

Other Techniques to Foil Session Hijackers

The following are techniques that can be used to further protect your web application from session hijackers.

1. *Form Input Validation* – The server should do fairly rigorous validation of all form input that comes from the clients. All the data in the GET and POST requests should be monitored to decrease the chances of successful HTML injection and cross-site scripting attacks.
2. *Session Time Out* – Users often share client machines, so it's important that the session time out after a certain period of inactivity. If the session never times out, this also gives the hijacker an unlimited amount of time to run his brute force attack.
3. *Token Regeneration* – Another way to limit the amount of time that the hijacker has for guessing a session ID before that session ID becomes useless is to regenerate the session token every once in a while. "The HTTP server can seamlessly expire and regenerate tokens to give an attacker a smaller window of time for replay exploitation of each legitimate token." [2]
4. *Re-authentication* – Along the same lines of our session ID discussion of distinguishing between secure and insecure parts of the site, we can treat critical parts of the site differently, requiring re-authentication. The result is that even if the hijacker gains control of the user session, he will not be able to do the more critical actions (e.g. in an online banking application, transferring money). This adds an extra layer of security against session hijackers.

5. *Brute Force Detection: Booby traps* – Making an effort to detect brute force session hijacking attempts is definitely a good way to protect your sessions. OWASP suggests using "booby trapped session tokens that never actually get assigned but will detect if an attacker is trying to brute force a range of tokens." [2] Once you have detected the session hijacking attempt, you can ban the originating IP address or lock out the account.

Determining Susceptibility: Helpful Tools

The following are some helpful tools to use to determine the susceptibility of your network and web applications to session hijackers:

MITM Proxy: Achilles and Paros

Achilles and Paros are man in the middle proxy tools. These tools "will intercept an HTTP session's data in either direction and give the user the ability to alter the data before transmission."[14] You can use these tools to determine how susceptible your network is to session hijackers setting up man in the middle situations. These two tools can intercept the packets between client and server and even negotiate separate SSL sessions, one with the server and one with the client so that it can forge data between them. Both programs are free for download.

Network Level: Juggernaut and Hunt

These two tools are like network sniffers that you can use to do TCP session hijacking. They can be set to listen for specific traffic and automate session hijacking attacks. The majority of the techniques mentioned in this paper, including ARP table modification and TCP resynchronization, are all built into Juggernaut and Hunt. They are useful tools for determining whether your network is vulnerable.

Application Level: SPI Dynamics Cookie Cruncher

One other tool I thought would be good to mention is the SPI Dynamics Cookie Cruncher. "SPI Dynamics' Cookie Cruncher tool analyzes cookies to determine how easy an attacker could predict or determine the value of a session ID generated by a server and delivered to a client via a cookie."[15] Basically, it's a good tool for figuring out how weak your cookies are against attacks by session hijackers.

Conclusion

Session hijacking remains a serious threat to networks and web applications on the web. This paper provided a general overview of how the nasty deed is done and how the information security engineer can protect networks and web applications from this threat. It is important to protect our session data at both the network and application levels. Although implementing all of the

countermeasures discussed here does not completely guarantee full immunity against session hijacking, it does raise the security bar and forces the session hijacker to come up with alternate and perhaps more complex methods of attack. It is a good idea to keep testing and monitoring our networks and applications to ensure that they will not be susceptible to the hijacker's tricks.

Works Cited Page

[1] Belani, Rohyt. "Basic Web Session Impersonation." Security Focus. 14 Apr. 2004. 20 Dec. 2004. <http://www.securityfocus.com/infocus/1774>

[2] Curphey, Mark, David Endler, William Hau, Steve Taylor, Tim Smith, Alex Russell, Gene McKenna, Richard Parke, Kevin McLaughlin, Nigel Tranter, Amit Klien, Dennis Groves, Izhar By-Gad, Sverre Huseby, Martin Eizner, Martin Eizner, and Roy McNamara "A Guide to Building Secure Web Applications." The Open Web Application Security Project. 11 Sept. 2002. 20 Dec. 2004. <http://prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download>

[3] Dittrich, Dave. "Session Hijack Script." Home Page. 8 Apr. 1999. 20 Dec. 2004. <http://staff.washington.edu/dittrich/talks/qsm-sec/script.html>

[4] Endler, David. "Brute-Force Exploitation of Web Application Session IDs." CGI Security. 1 Nov. 2001. 20 Dec. 2004.

<http://www.cgisecurity.com/lib/SessionIDs.pdf>
[5] Imperva. “Glossary: What is Session Hijacking?” Imperva: Total Application Security. Accessed: 20 Dec. 2004. <http://www.imperva.com/application_defense_center/glossary/session_hijacking.html>
[6] Internet Security Systems. “Session Hijacking.” Internet Security Systems. Accessed: 20 Dec. 2004. <http://www.iss.net/security_center/advice/Exploits/TCP/session_hijacking/default.htm>
[7] Joncheray, Laurent. “Simple Active Attack Against TCP.” Church of the Swimming Elephant. June 1995. 20 Dec. 2004. <http://www.cotse.com/texts/iphijack.txt>
[8] Lam, Kevin, David LeBlanc, and Ben Smith. “Hacking: Fight Back: Theft On The Web: Prevent Session Hijacking.” Microsoft TechNet. Winter 2005. 1 Jan. 2005. <http://www.microsoft.com/technet/technetmag/issues/2005/01/sessionhijacking/default.aspx?pf=true>
[9] Morana, Marco. “Make It and Break It: Preventing Session Hijacking and Cookie Manipulation.” Secure Enterprise. 23 Nov. 2004. 20 Dec. 2004. <http://nwc.securitypipeline.com/howto/53701241>
[10] Ollman, Gunter. “Web Session Management: Best Practices in Managing HTTP Based Client Sessions.” Technical Info: Making Sense of Security. Accessed: 20 Dec. 2004. <http://www.technicalinfo.net/papers/WebBasedSessionManagement.html>
[11] Paros MITM Proxy. 2004. 20 Dec. 2004 <http://www.parosproxy.org/>
[12] Prosise, Jeff. “Wicked Code: Foiling Session Hijacking Attempts.” MSDN Magazine. Aug. 2004. 20 Dec. 2004. <http://msdn.microsoft.com/msdnmag/issues/04/08/WickedCode/>
[13] SecuriTeam. “Juggernaut, a session hijacking tool.” SecuriTeam. 28 Jan. 1999. 20 Dec. 2004. <http://www.securiteam.com/tools/2NUQBSAQ0C.html>
[14] SecuriTeam. “Achilles tests the security of web applications.” SecuriTeam. 22 Dec. 2000. 20 Dec. 2004. <http://www.securiteam.com/tools/6L00R200KA.html>
[15] SPI Dynamics (Cookie Cruncher). 2004. 20 Dec. 2004. <http://www.spidynamics.com/products/security/toolkit/ccruncher.html>
[16] WebKreator. “Advanced Session Security.” WebKreator. 20 Feb. 2002. 20 Dec. 2004. <http://www.webkreator.com/cms/view.php/1685.html>
[17] Webopedia: Online Computer Dictionary for Computer and Internet Terms and Definitions. 2004. 20 Dec. 2004. <http://www.webopedia.com/>

Last Updated: October 21st, 2017

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS San Diego 2017	San Diego, CAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Seattle 2017	Seattle, WAUS	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Gulf Region 2017	Dubai, AE	Nov 04, 2017 - Nov 16, 2017	Live Event
SANS Milan November 2017	Milan, IT	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Amsterdam 2017	Amsterdam, NL	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Miami 2017	Miami, FLUS	Nov 06, 2017 - Nov 11, 2017	Live Event
SANS Paris November 2017	Paris, FR	Nov 13, 2017 - Nov 18, 2017	Live Event
Pen Test Hackfest Summit & Training 2017	Bethesda, MDUS	Nov 13, 2017 - Nov 20, 2017	Live Event
SANS Sydney 2017	Sydney, AU	Nov 13, 2017 - Nov 25, 2017	Live Event
GridEx IV 2017	Online,	Nov 15, 2017 - Nov 16, 2017	Live Event
SANS San Francisco Winter 2017	San Francisco, CAUS	Nov 27, 2017 - Dec 02, 2017	Live Event
SANS London November 2017	London, GB	Nov 27, 2017 - Dec 02, 2017	Live Event
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZUS	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Khobar 2017	Khobar, SA	Dec 02, 2017 - Dec 07, 2017	Live Event
SANS Austin Winter 2017	Austin, TXUS	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Munich December 2017	Munich, DE	Dec 04, 2017 - Dec 09, 2017	Live Event
European Security Awareness Summit & Training 2017	London, GB	Dec 04, 2017 - Dec 07, 2017	Live Event
SANS Bangalore 2017	Bangalore, IN	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Frankfurt 2017	Frankfurt, DE	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DCUS	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LAUS	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS SEC460: Enterprise Threat Beta	San Diego, CAUS	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Amsterdam January 2018	Amsterdam, NL	Jan 15, 2018 - Jan 20, 2018	Live Event
Northern VA Winter - Reston 2018	Reston, VAUS	Jan 15, 2018 - Jan 20, 2018	Live Event
SEC599: Defeat Advanced Adversaries	San Francisco, CAUS	Jan 15, 2018 - Jan 20, 2018	Live Event
SANS Berlin 2017	OnlineDE	Oct 23, 2017 - Oct 28, 2017	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced