© SANS Institute 2021

507.5

Auditing Web Applications

August 10, 2021

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

August 10, 2021

Technet24

© SANS Institute 2021

© 2021 Risenhoover Consulting, Inc. All rights reserved to Risenhoover Consulting, Inc. and/or SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE “USER”) AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE,USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, USER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree, User may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There’s an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP® and PMBOK® are registered trademarks of PMI.

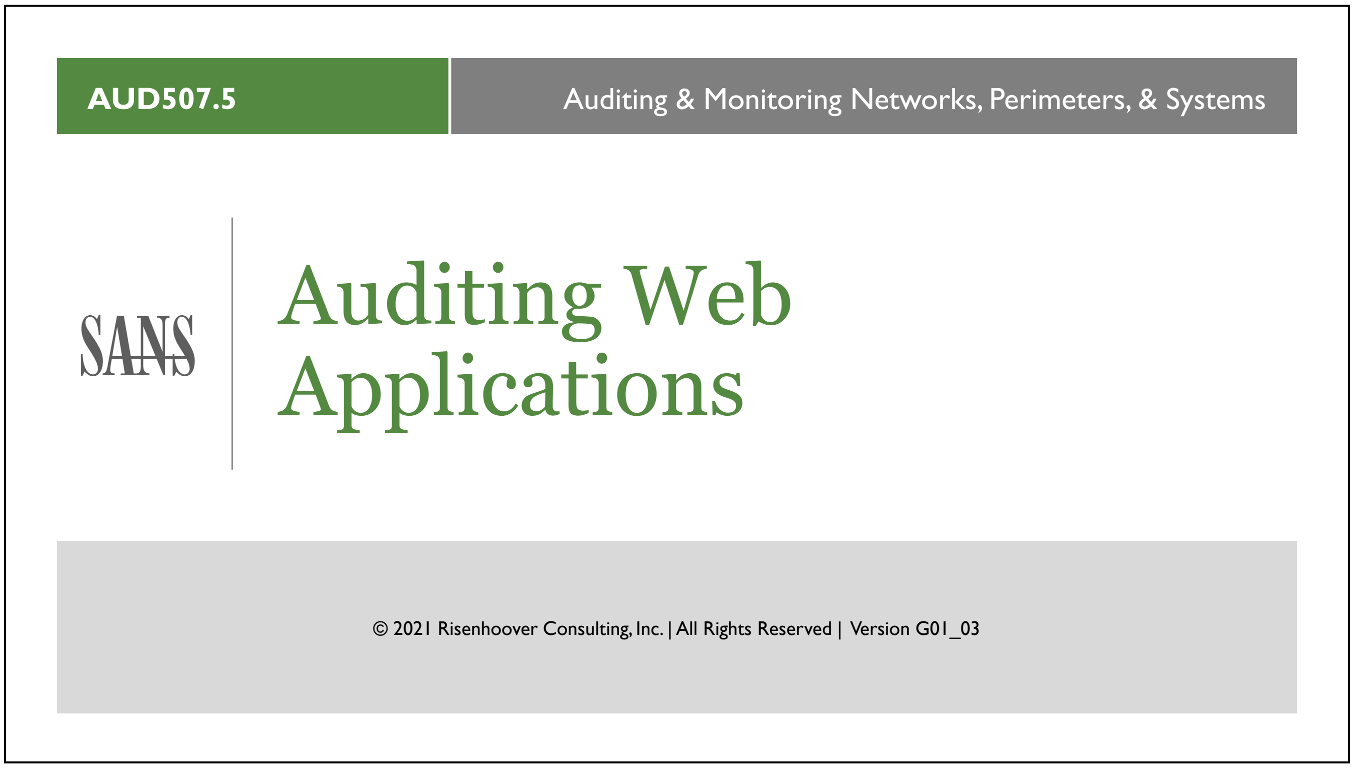
SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

August 10, 2021

© SANS Institute 2021



AUD507.5

Auditing & Monitoring Networks, Perimeters, & Systems

Auditing Web Applications

© 2021 Risenhoover Consulting, Inc. | All Rights Reserved | Version G01_03

Welcome to Section Five of the SANS AUD507 course! This course is written, maintained, and frequently taught by Clay Risenhoover. I am always looking for ways to improve this courseware. If you have questions or suggestions for how to improve the course, or if you need any additional materials referenced during the class, please let me know. If you find errors or inaccuracies in the course books, I encourage you to pass those on to me. You can email me at clay@risenhooverconsulting.com. Please put either "SANS" or "AUD507" in the subject line, to ensure I see the email.

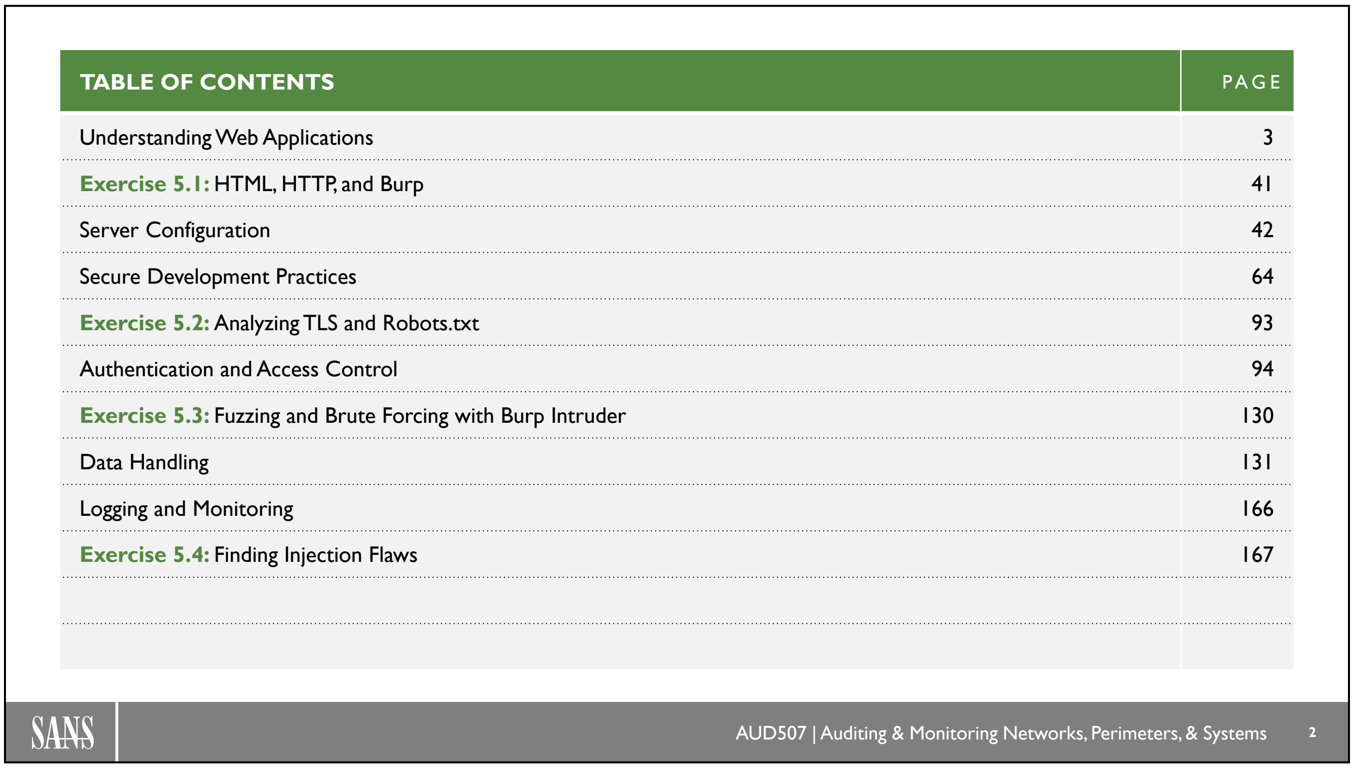
The entire content of this and every other volume in this course is © 2021 Risenhoover Consulting, Inc.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 2

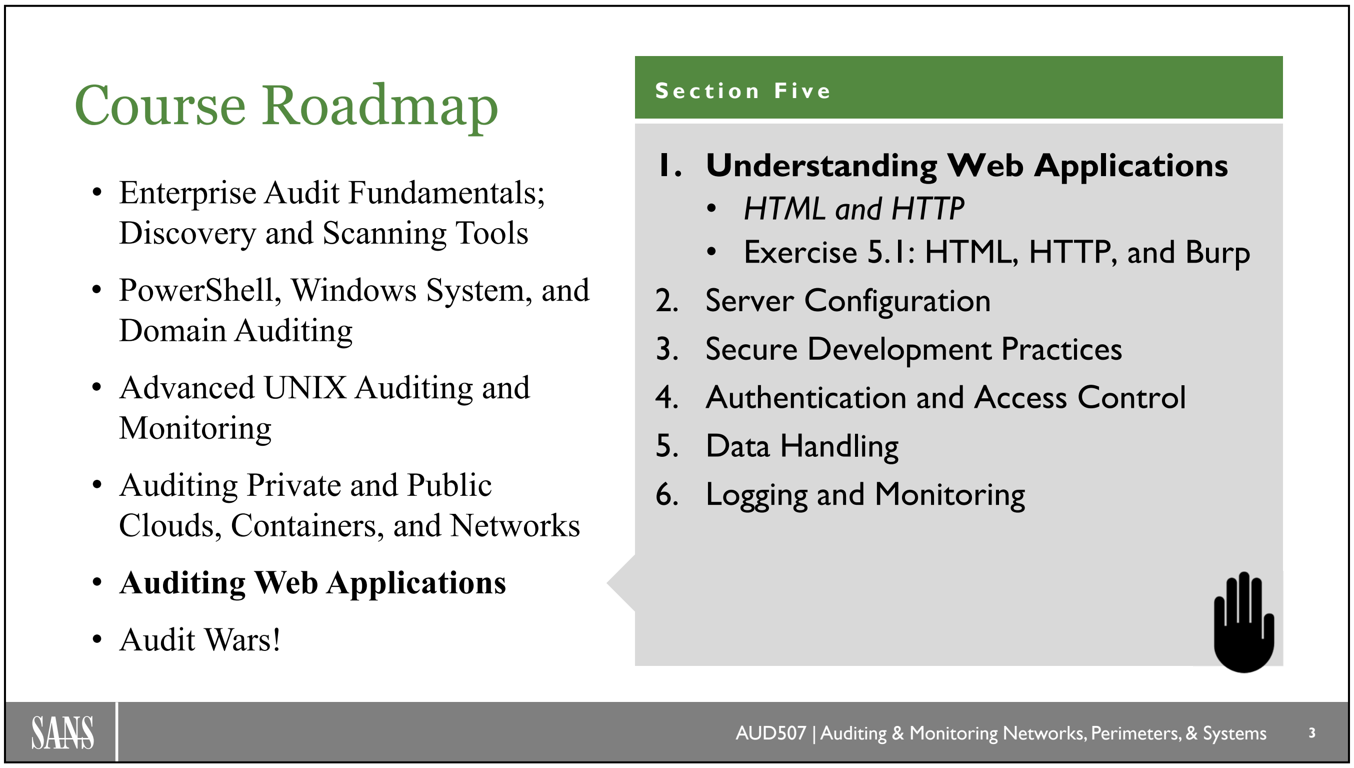
This page intentionally left blank.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Course Roadmap

- Enterprise Audit Fundamentals; Discovery and Scanning Tools
- PowerShell, Windows System, and Domain Auditing
- Advanced UNIX Auditing and Monitoring
- Auditing Private and Public Clouds, Containers, and Networks
- **Auditing Web Applications**
- Audit Wars!

Section Five

1. **Understanding Web Applications**
   - *HTML and HTTP*
   - Exercise 5.1: HTML, HTTP, and Burp
2. Server Configuration
3. Secure Development Practices
4. Authentication and Access Control
5. Data Handling
6. Logging and Monitoring

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 3

Welcome to the Web Application Auditing portion of the SANS Audit 507 track. We hope you'll find this particular section to be as interesting and exciting as many thousands of previous attendees have. A fair number have written in to state that they never look at their web traffic the same way again. Some have even taken the tools that we demonstrate and use them to analyze all of their web traffic to help them decide who they want to do business with.

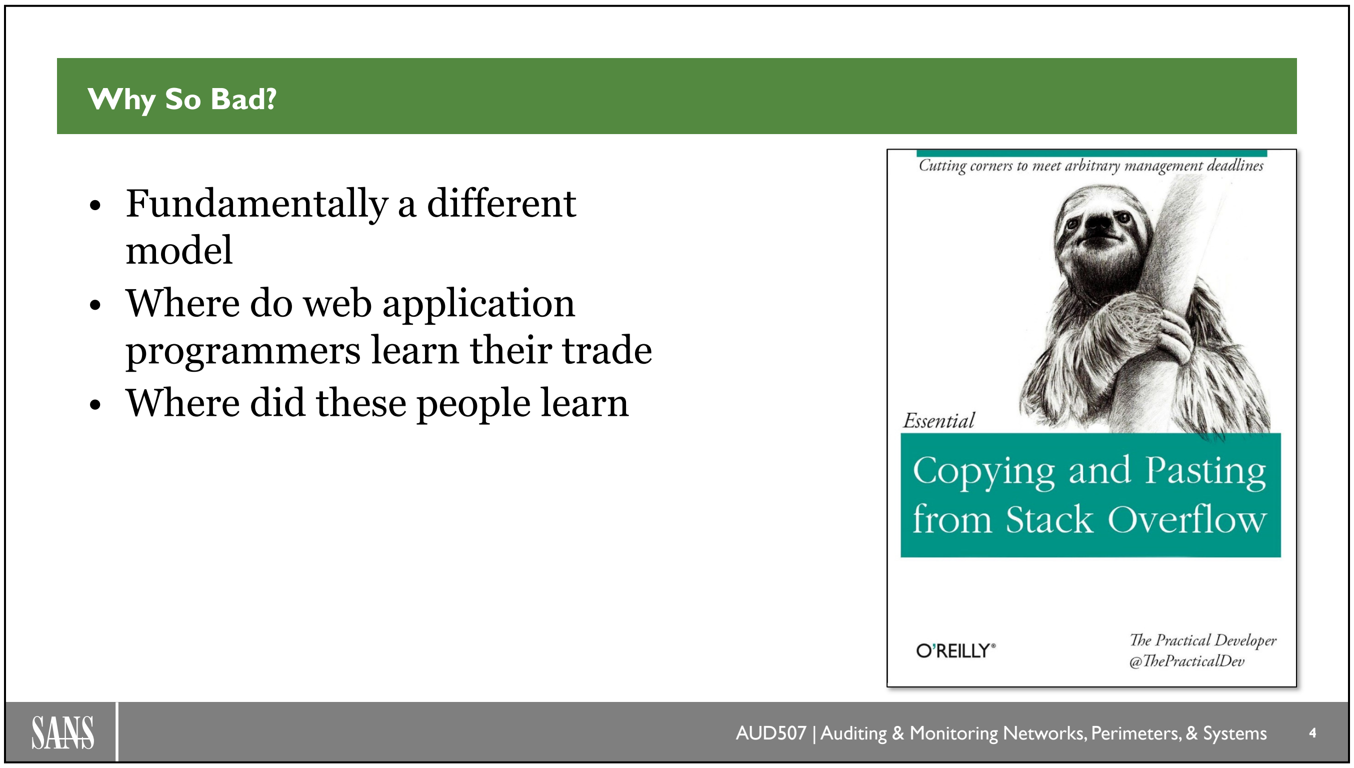
Unlike the topics that we cover over the rest of the course (operating systems), many new concepts are discussed in this section. While everyone has used the web, relatively few actually understand how web applications *should* be created, so we will start with some basics.

August 10, 2021

August 10, 2021

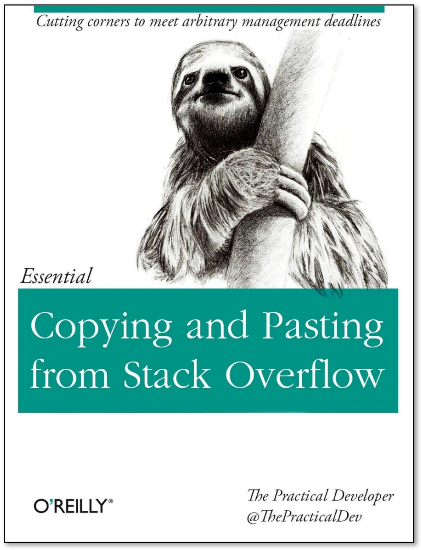
Technet24

© SANS Institute 2021



Why So Bad?

- Fundamentally a different model
- Where do web application programmers learn their trade
- Where did these people learn


Cutting corners to meet arbitrary management deadlines
Essential
Copying and Pasting from Stack Overflow
O'REILLY
The Practical Developer
@ThePracticalDev

Some of the more fundamental reasons for the problems that we face with web applications have to do with basic function and training. I have been hired several times to work with large organizations to oversee their application development team in the creation of a web application. Every time, I begin by teaching their programmers how to write secure web applications. What most programmers have the hardest time understanding is that the web client has far more control over the application than is the case in almost any other form of application programming. We'll see why this is the case as we go on this morning.

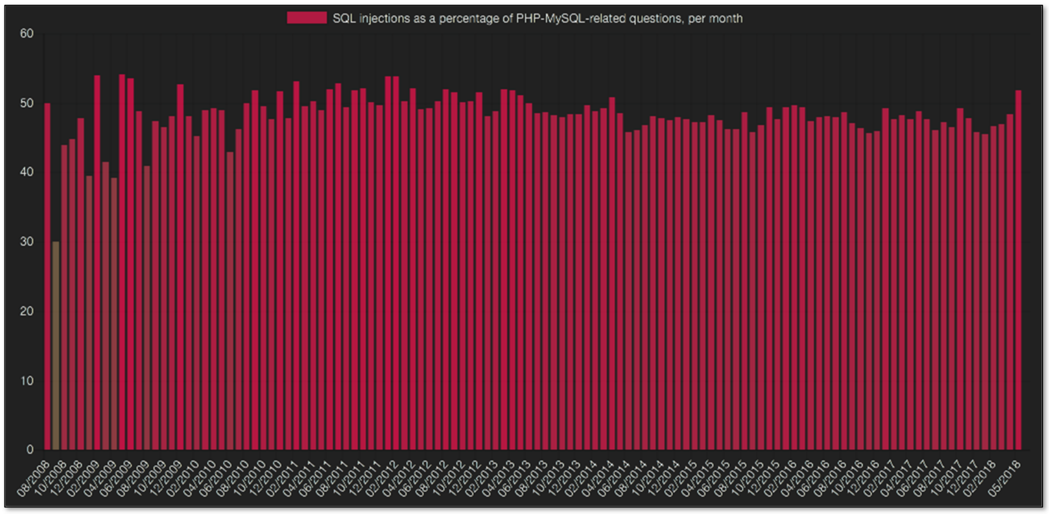
When it comes to training, consider this: Where did your web application programmers learn how to do what they do? Did they learn on their own? Perhaps in college or a training seminar? Where did the person who taught them learn? Somewhere along the line, you will come to a point in which someone essentially taught themselves. We're not saying that you can't teach yourself effectively, but any misconceptions or bad practice that this person had will tend to rub off on the people that they teach.

There's an even deeper root cause, though. If you were going to teach yourself, how would you do it? A reasonable answer is to review sample code and other applications. The problem in this space is that almost every example application distributed with web application frameworks has enormous vulnerabilities in it! Even if you were to buy a book on web application programming, these will frequently either sacrifice discussions of security in order to simplify the presentation of the topics or reveal that the author is completely unaware of how to properly secure an application. Even websites offering specialized coding advice and assistance are found to have a large amount of bad advice, at least from a security point of view.

© 2021 Risenhoover Consulting, Inc.
August 10, 2021

© SANS Institute 2021

StackOverflow: Percentage of Questions with SQL Vulnerabilities



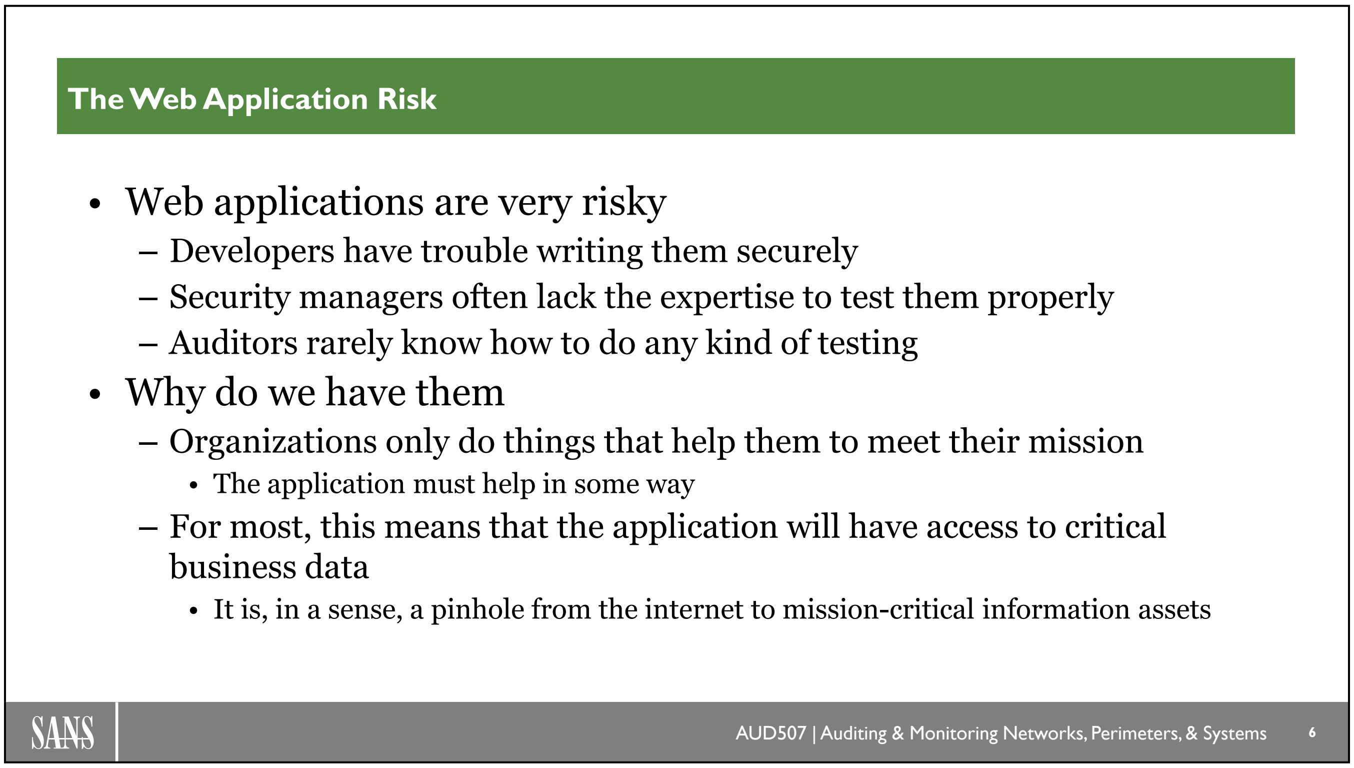
Just as proof of what we're telling you, have a look at this chart. Here we are looking at just one programming language (PHP) and just one issue (SQL injection). We can say that SQL injection in PHP code is extremely well understood. It is so well understood that someone created a GitHub project that tracked *questions with SQL injection flaws in them on StackOverflow* for ten consecutive years. In 2018, *about half* of the PHP questions posed on StackOverflow that include SQL code in them have very easy-to-find (and exploit) SQL injection flaws in them!

You might hear that and say, "well, ok, but those are the questions. Surely the answers correct these issues, right?" Absolutely not—in fact, we often see that the answers add NEW vulnerabilities into the code.

August 10, 2021

August 10, 2021
Technet24

© SANS Institute 2021



The Web Application Risk

- Web applications are very risky
  - Developers have trouble writing them securely
  - Security managers often lack the expertise to test them properly
  - Auditors rarely know how to do any kind of testing
- Why do we have them
  - Organizations only do things that help them to meet their mission
    - The application must help in some way
  - For most, this means that the application will have access to critical business data
    - It is, in a sense, a pinhole from the internet to mission-critical information assets

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 6

Something that can help you to appreciate the criticality of the risk related to web applications is to consider them in the context of the business. Most publicly exposed systems that organizations have are both well hardened and have (or provide access to) very little sensitive information. Web applications, while they should be well written and hardened, are quite different in that they are specifically intended to assist the organization to make highly sensitive information accessible over the internet. You could really view the web application as a proxy to your core database tables.

When we consider an application in this light, ensuring that it is well written and secure becomes one of the most urgent and important tasks for a security and audit team. Even so, our experience is that many security managers have no idea what the various web application findings described by vulnerability assessment tools and penetration testers mean. If you don't understand what the issue is, how can you possibly understand how serious the risk is? We see security managers elect to ignore findings that testers mark as "critical" because the programmers assure them that the tester is wrong, and the manager really doesn't understand what the issue is.

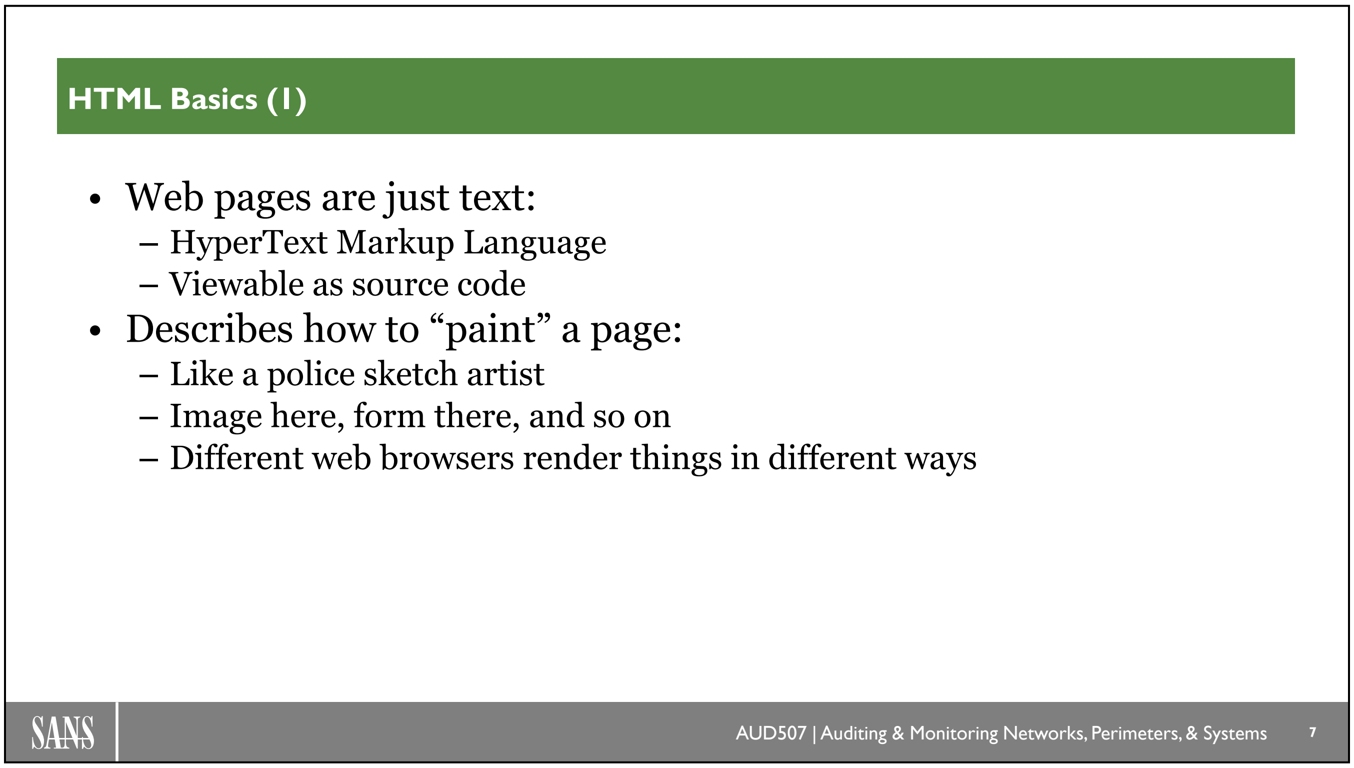
Similarly, most auditors have no idea how to perform any kind of validation testing. Certainly, auditors know (or should know) to ask about development process, deployment standards, patching, and testing requirements; what we're saying is that the auditor will often not have any of the technical knowledge necessary to validate anything that is claimed by the development team. It is our aim to change your relationship to this problem with this section of the course.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



HTML Basics (1)

- Web pages are just text:
  - HyperText Markup Language
  - Viewable as source code
- Describes how to "paint" a page:
  - Like a police sketch artist
  - Image here, form there, and so on
  - Different web browsers render things in different ways

Web application auditing tends to be easier than most other forms of application auditing because almost everything that we need to do our testing is presented to us by the web server as text. When you look at a webpage, it can appear quite complex in terms of layout and design, but fundamentally few pieces of the application actually matter for testing, and everything that we're concerned about is text.

The reason that this is true is that HTML, the language used to create most webpages today, is simply text. Not only that, the source code for any webpage is only a few mouse clicks away because we can view the source code for any webpage that our browser can display!

Essentially, HyperText Markup Language (HTML) is a language that describes how a page should look. It is then up to your browser to interpret that language to paint the page correctly. This is the reason that you will find that different web browsers may display the same page somewhat differently. You might think of this as several artists being asked to render the same picture.

August 10, 2021

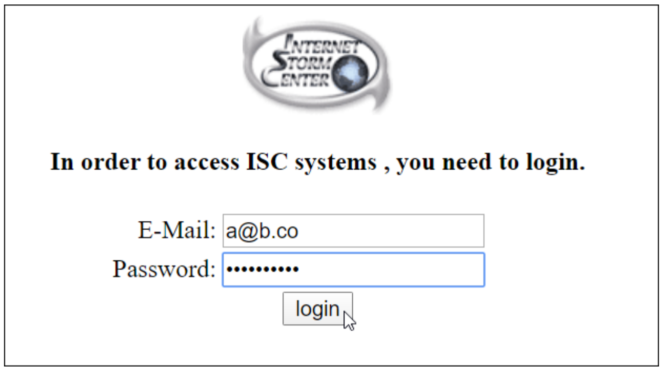
August 10, 2021

Technet24

© SANS Institute 2021

HTML Basics (2)

- Consider a VERY simple web page:


In order to access ISC systems , you need to login.

E-Mail: a@b.co
Password: ••••••••••

Take a look at this simple web page. If contains an image - the SANS Internet Storm Center logo, some formatted text, and a web form that allows the user to input a username and password before clicking the "login" button.

What does the HTML that describes this page look like?

August 10, 2021

August 10, 2021

© SANS Institute 2021

HTML Basics (3)

```
<html><head><title>Sample Web Page</title></head>
<body><img src="https://isc.sans.edu/img/logo.png"><br />
<h4>In order to access ISC systems , you need to login.</h4>
  <form method="post" name="loginform" action="login.html">
    <input type="hidden" name="token" value="21b47cf795def6e561b4671befcea">
       <table width="300"><tbody><tr><td align="right">E-Mail:</td>
      <td><input type="text" name="email"></td></tr>
      <tr><td align="right">Password:</td>
        <td><input autocomplete="off" type="password" name="pw"></td></tr>
      <tr><td colspan="2" align="center">
       <input type="submit" name="submit" value="login"></td></tr>
    </tbody></table>
</form>
</body></html>
```

The HTML text contains “tags” which tell the browser how to interpret and display the text in the page. The <h4> and </h4> tags tell the browser to treat the text between them as a heading and format it accordingly. How the heading is formatted will vary based on the browser and any styles which have been applied to the page.

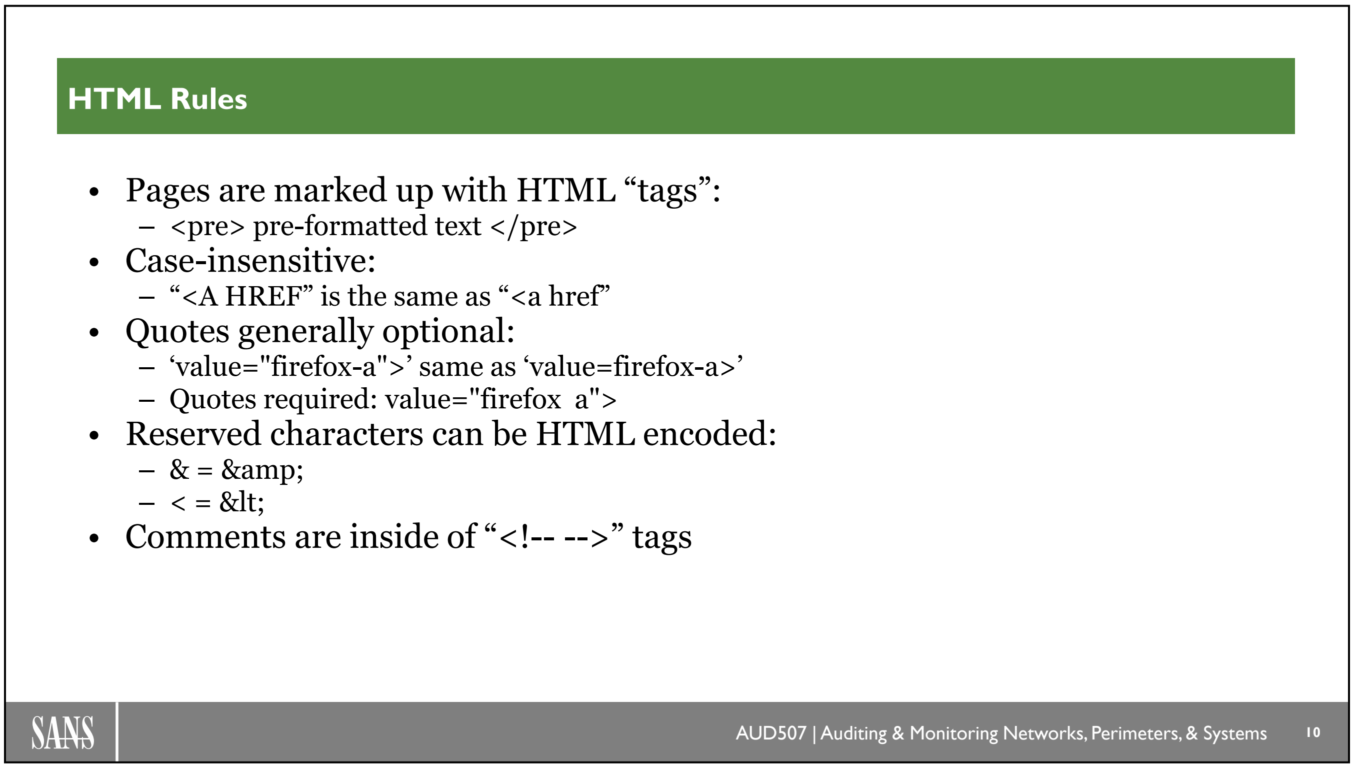
There is also a form which is used to take input from the user (or return information embedded in the source code, like the “hidden” form field named “token” in the example) and return it to the server. Because the form takes input from the user, you should pay special attention to it. Any time you see a form in a web application, remember that the user can manipulate the input they send to try to compromise the application.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



HTML Rules

- Pages are marked up with HTML "tags":
  - <pre> pre-formatted text </pre>
- Case-insensitive:
  - "<A HREF" is the same as "<a href"
- Quotes generally optional:
  - 'value="firefox-a">' same as 'value=firefox-a>'
  - Quotes required: value="firefox a">
- Reserved characters can be HTML encoded:
  - & = &amp;
  - < = &lt;
- Comments are inside of "<!-- -->" tags

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 10

This concept of using marks or "tags" is fundamental to HTML. It is important to understand that almost all tags also require some sort of closing tag to identify when the formatting instruction ends. For instance, if a <center> mark were used to center some text, how would the web browser know when to stop centering text? The answer is that there is a closing tag, too. The closing tags always mirror the opening tag, they simply add a forward slash. This means that for the <pre> tag there would be a corresponding </pre> tag. You can think of these opening and closing tags like parentheses. If you open a parenthetical expression (like this), the rules of syntax tell you that there must be a closing parenthesis.

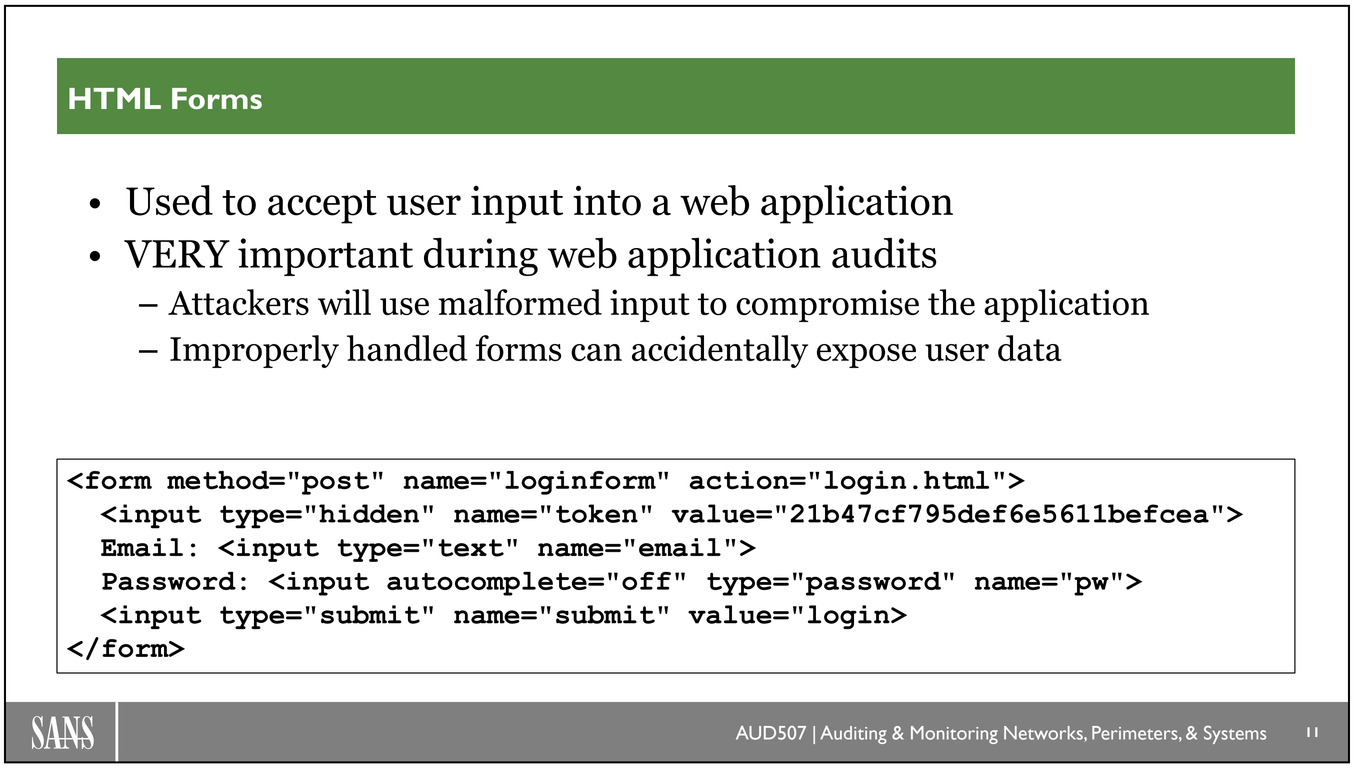
A few other things to know are that HTML is case-insensitive. Also, quotation marks inside of HTML tags are generally optional unless you need to use a space within an expression. HTML also has some special characters, such as the < and >. What if you want to use one of these? You can mark those characters using &gt; and &lt;. These are examples of HTML encoding.

An important tag to know about is the comment tag. Comments are marked with <!-- -->. Nothing within these will be displayed within the rendered page.

August 10, 2021

August 10, 2021

© SANS Institute 2021



HTML Forms

- Used to accept user input into a web application
- VERY important during web application audits
  - Attackers will use malformed input to compromise the application
  - Improperly handled forms can accidentally expose user data

```
<form method="post" name="loginform" action="login.html">
  <input type="hidden" name="token" value="21b47cf795def6e5611befcea">
  Email: <input type="text" name="email">
  Password: <input autocomplete="off" type="password" name="pw">
  <input type="submit" name="submit" value="login>
</form>
```

Any time a web application handles user input, there is additional risk. It’s important that as a web application auditor you learn to recognize input forms when you see them, and that you take the time to analyze their security effects on the application.

One important thing to note is the HTTP method which is being used to send the user input to the server. In this example, the form is using the HTTP “POST” method. We’ll discuss HTTP methods (or verbs) later in this section of the material.

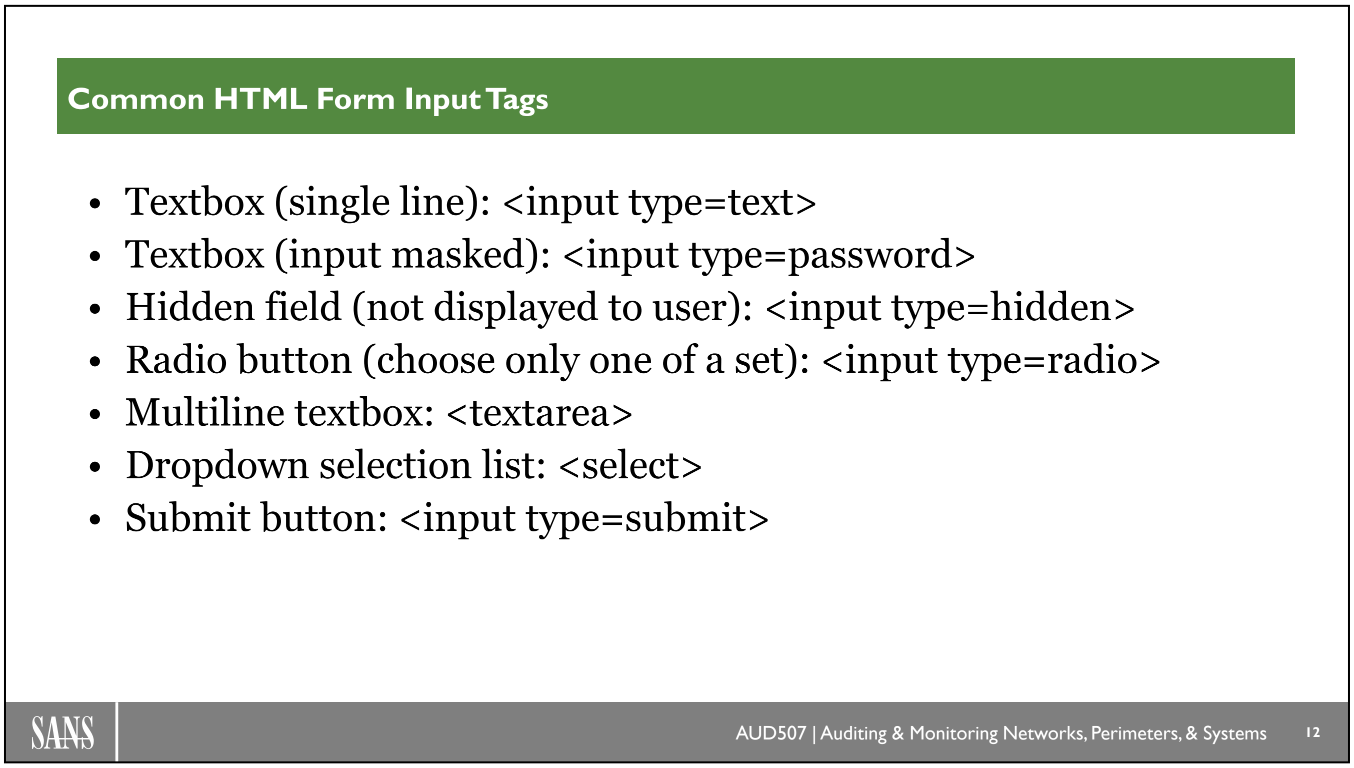
It’s also important to understand the types of HTML input fields you might see in the web applications you audit.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Common HTML Form Input Tags

- Textbox (single line): <input type=text>
- Textbox (input masked): <input type=password>
- Hidden field (not displayed to user): <input type=hidden>
- Radio button (choose only one of a set): <input type=radio>
- Multiline textbox: <textarea>
- Dropdown selection list: <select>
- Submit button: <input type=submit>

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 12

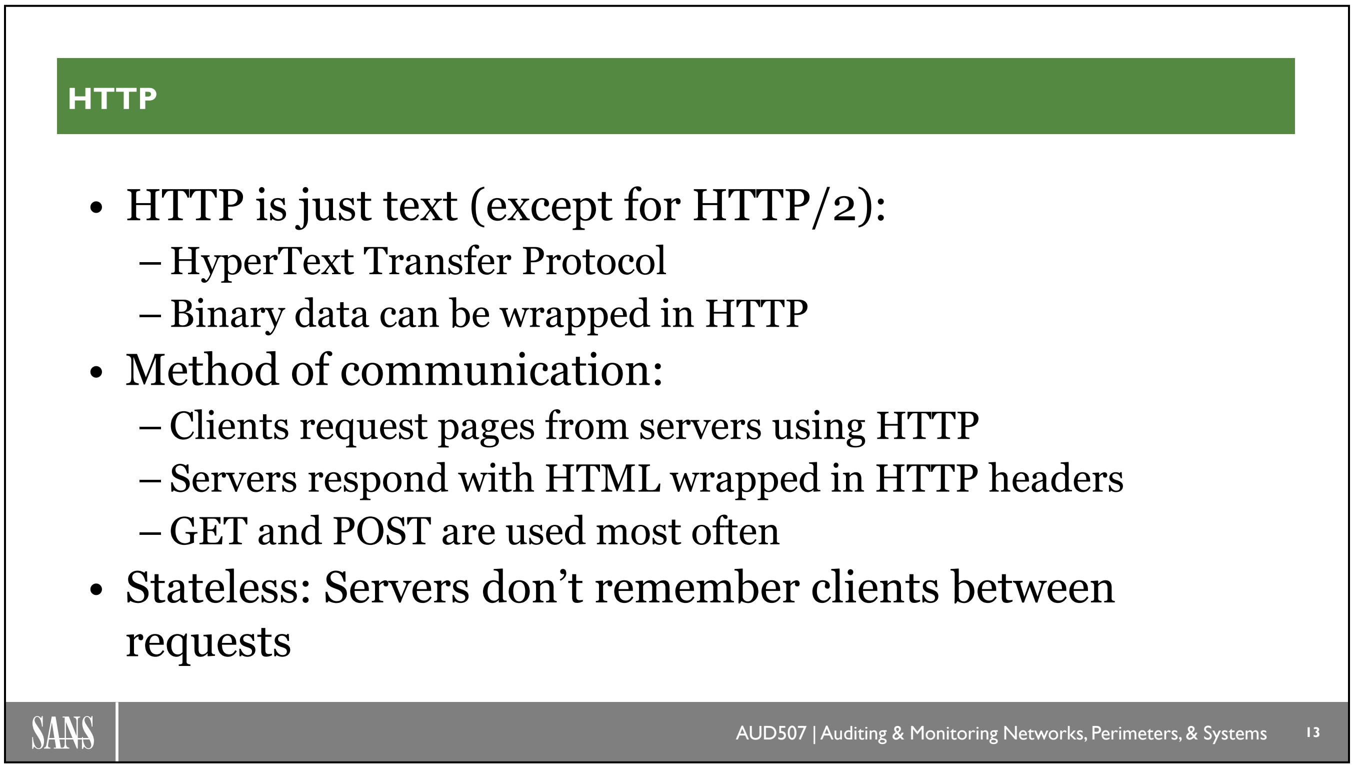
This slide includes a (non-comprehensive) list of some of the common HTML form input types. Notice that while most of them have tags which begin with the word “input,” not all do. Text areas and select boxes are notable exceptions.

Hidden form fields are interesting as well. Why would a developer have a field in a form and then hide it from the user? Aren’t forms intended specifically to solicit input from users? Yes, but it turns out that sometimes the developer needs a way of passing input back into a web application to maintain “state.” We’ll discuss more about the stateless nature of the HTTP protocol later in the section. For right now, understand that while a developer may think that hidden form fields cannot be manipulated by the end user, this is not really true. Users can employ a number of tools and techniques to manipulate the input into your application—even if you think it’s hidden from them.

August 10, 2021

August 10, 2021

© SANS Institute 2021



HTTP

- HTTP is just text (except for HTTP/2):
  - HyperText Transfer Protocol
  - Binary data can be wrapped in HTTP
- Method of communication:
  - Clients request pages from servers using HTTP
  - Servers respond with HTML wrapped in HTTP headers
  - GET and POST are used most often
- Stateless: Servers don't remember clients between requests

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 13

HTML and HTTP are not the same thing. HTTP, or HyperText Transfer Protocol, is the communication mechanism used to transfer hypertext documents. In reality, the protocol can also be used to transfer binary data or just about anything else (like images), but its primary role is to allow browsers to speak to web servers. When a web browser sends a request to a web server, the client's request is made using HTTP. When the server responds, it answers back by sending HTML embedded within HTTP headers. In other words, HTTP is the transport mechanism.

Within HTTP there are a number of different request types. Of the many that exist, there are only two that we care about for most web applications, and these two are our focus. There is, in reality, a third that you may encounter: HEAD. Head, however, is used only to determine if a page has changed. In other words, the web browser is trying to determine whether it should try to request the entire page.

GET and POST are the two most common methods that are used to send data to a web server and to retrieve data from a web server. Even though they serve essentially the same function, the specifics of how they function are quite different.

GET takes all the values entered in the form and appends them to the URL listed in the action attribute of the <form> tag. GET is easy to use, but it does have some limitations and security issues. We'll discuss the security issues later, but for now let's just give you the limitation: You can't send more than 255 characters in the URL with a GET.

POST essentially performs the same task that GET does with two major exceptions. The first is that there is no hard limit on how much data can be sent with a POST. The second is that the form values are not included in the URL; they are included in the body of the request.

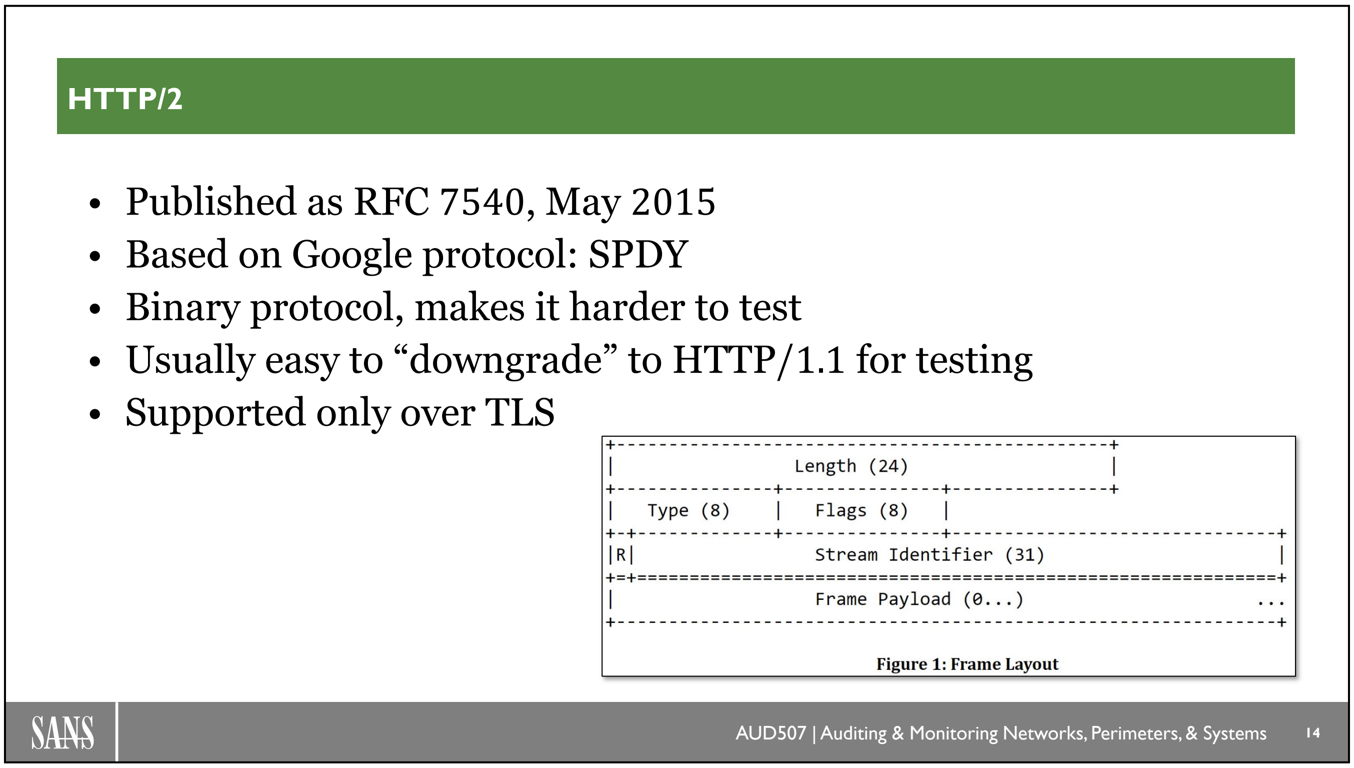
The fact the HTTP is stateless – a server does not remember a client between requests – makes for some interesting problems that we'll work through as the section goes on.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



HTTP/2

- Published as RFC 7540, May 2015
- Based on Google protocol: SPDY
- Binary protocol, makes it harder to test
- Usually easy to "downgrade" to HTTP/1.1 for testing
- Supported only over TLS

```
+-----------------------------------------------+
|                 Length (24)                   |
+---------------+---------------+---------------+
|   Type (8)    |   Flags (8)   |
+-+-------------+---------------+-------------------------------+
|R|                 Stream Identifier (31)                      |
+=+=============================================================+
|                   Frame Payload (0...)                      ...
+---------------------------------------------------------------+
```

Figure 1: Frame Layout

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 14

HTTP/2 is the latest version of the HTTP protocol. Unlike previous versions, HTTP/2 is binary, using a format that looks more like the TCP header than traditional HTTP. HTTP/2 is built on a Google-developed protocol named "SPDY" that was later adopted by the industry at large.

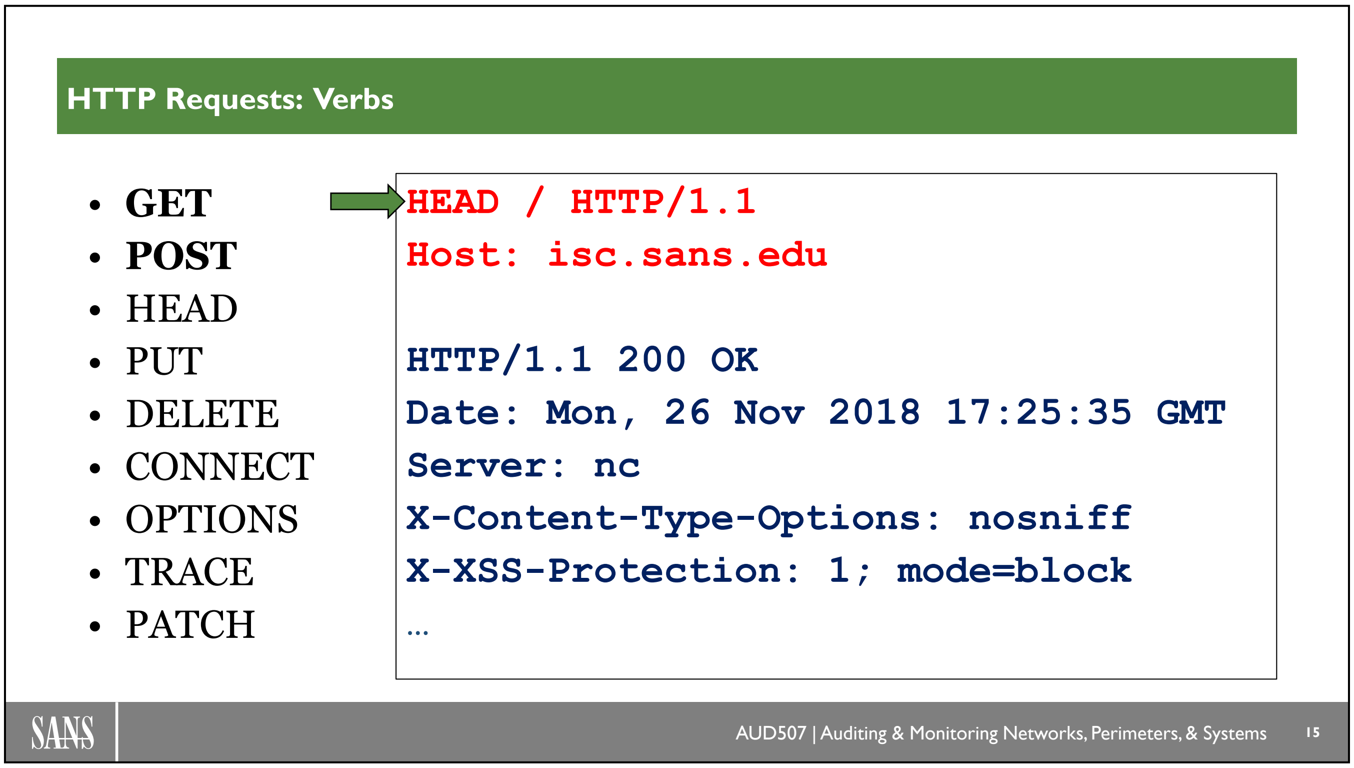
Many of our testing tools, like Burp, don't yet have good support for HTTP/2. The solution is usually to allow the tool to downgrade to HTTP version 1.1, which is well supported by most testing tools.

While the protocol does not explicitly require the use of HTTPS for HTTP/2, all major browser vendors have implemented HTTP/2 to run only over TLS.

August 10, 2021

August 10, 2021

© SANS Institute 2021



HTTP Requests: Verbs

- **GET**
- **POST**
- HEAD
- PUT
- DELETE
- CONNECT
- OPTIONS
- TRACE
- PATCH

```
HEAD / HTTP/1.1
Host: isc.sans.edu

HTTP/1.1 200 OK
Date: Mon, 26 Nov 2018 17:25:35 GMT
Server: nc
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
…
```

HTTP clients, such as web browsers, use verbs to tell HTTP servers what kind of request they are making. The verbs which will attract most of your attention as a web application auditor will be GET and POST. These verbs are used for most of the requests which you will analyze.

The other verbs listed are commonly used in web services, single-page applications, and HTTP services like WebDAV. We will discuss these as we work through the material.

August 10, 2021

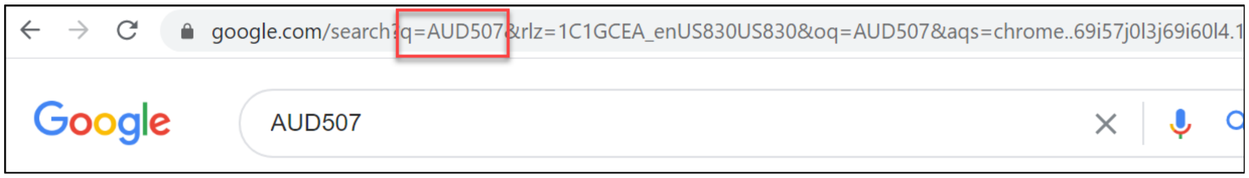
August 10, 2021

Technet24

© SANS Institute 2021

HTTP Requests: GET

- Google uses GET:
  - Enter your search term:
  - When search completes, look at the URL:



google.com/search?q=AUD507&rlz=1C1GCEA_enUS830US830&oq=AUD507&aqs=chrome..69i57j0l3j69i60l4.1

Okay, so we told you the theory. If you're still confused, don't worry; let's look at some pictures that should clear things up. Notice in the slide that we have a shot of the Google search page. The Google search page uses GET to send your requests. That means that the form definition includes <form method=get>.

To get an idea of what this looks like when you submit a request, look at the URL screenshot from a browser at the bottom of the slide. Notice that the search term that we entered, **AUD507**, shows up in the URL as q=AUD507. You may also notice that there are a number of other items in the URL. Where did these come from? No doubt these were hidden form elements in the search page.

There are a few more details to notice here. The URL, if you've never taken a close look at anything more than the part where the host is defined, might look a bit complicated. Let's see how to decode the URL.

The HTTP standards require that any form elements sent in a request to a web server using a GET be included in the URL, but how can we add things to the URL without confusing the web browser? How will it know where to go? The key is the question mark. Everything that appears in the URL before the question mark identifies the site to connect to. The question mark is like a fulcrum around which the URL pivots. Everything following the question mark is input to the web application being contacted - known as the "query string". In this case, that input is the form data that is being submitted.

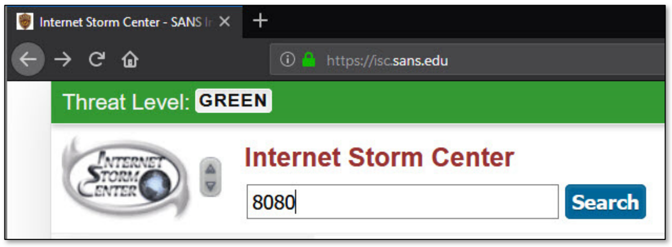
A few other characters are used to help describe this data. First, the equals sign is used to assign names to the values. These names are the names included in the element definition in the web form. If there are multiple elements, the various elements will be separated by the ampersand. Finally, spaces in values are converted into the plus sign. This is sometimes defined as HTTP encoding. Any other reserved characters would be converted to URL encoding or hexadecimal encoding, which means they are given in hexadecimal.

August 10, 2021

© SANS Institute 2021

HTTP Requests: POST

- ISC uses POST:
  - To see a post, we need to intercept the request
  - Burp Proxy



```
POST /search.html HTTP/1.1
Host: isc.sans.edu
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:64.0)
Gecko/20100101 Firefox/64.0
Content-Type: application/x-www-form-urlencoded
Referer: https://isc.sans.edu/

q=8080&token=
```

$\left.\right\}$ Headers

$\left.\right\}$ Body

Contrast the GET request with the POST. In this slide, we're looking at the SANS Internet Storm Center search form. The form, in this case, uses the POST. The request is displayed at the bottom of the page. This is the HTTP that was sent from the client to the server. We've used a special tool to intercept that data, and we'll show you how to do that in just a little while.

For now, notice that the first line in that request is a POST with a URL. Following the POST, we have a variety of HTTP headers, including information about the web browser (User-Agent), the name of the host we're connecting to (Host), referrer information (Referer), and so on. At the bottom, though, we find a series of parameters that looks like the format used with a GET! In fact, the format is identical.

So, what's the difference? In the GET, those parameters appear in the URL; in a POST the parameters are in the body of the HTTP request. Why this is important may not be obvious now, but don't worry, we'll tell you why it's important later!

An interesting side point is that the URL in a POST can still contain additional parameters that will be passed to the web application. This is a valuable technique for web application programmers that we'll look at later when we consider maintaining sessions.

Although we're not going to spend a great deal of time with all the various HTTP headers, it is worthwhile taking note of them when you see them. They can frequently give you a lot of valuable information including the type and version of the browser in use, any plugins installed for handling various media types, and where the browser was before it came to the current site.

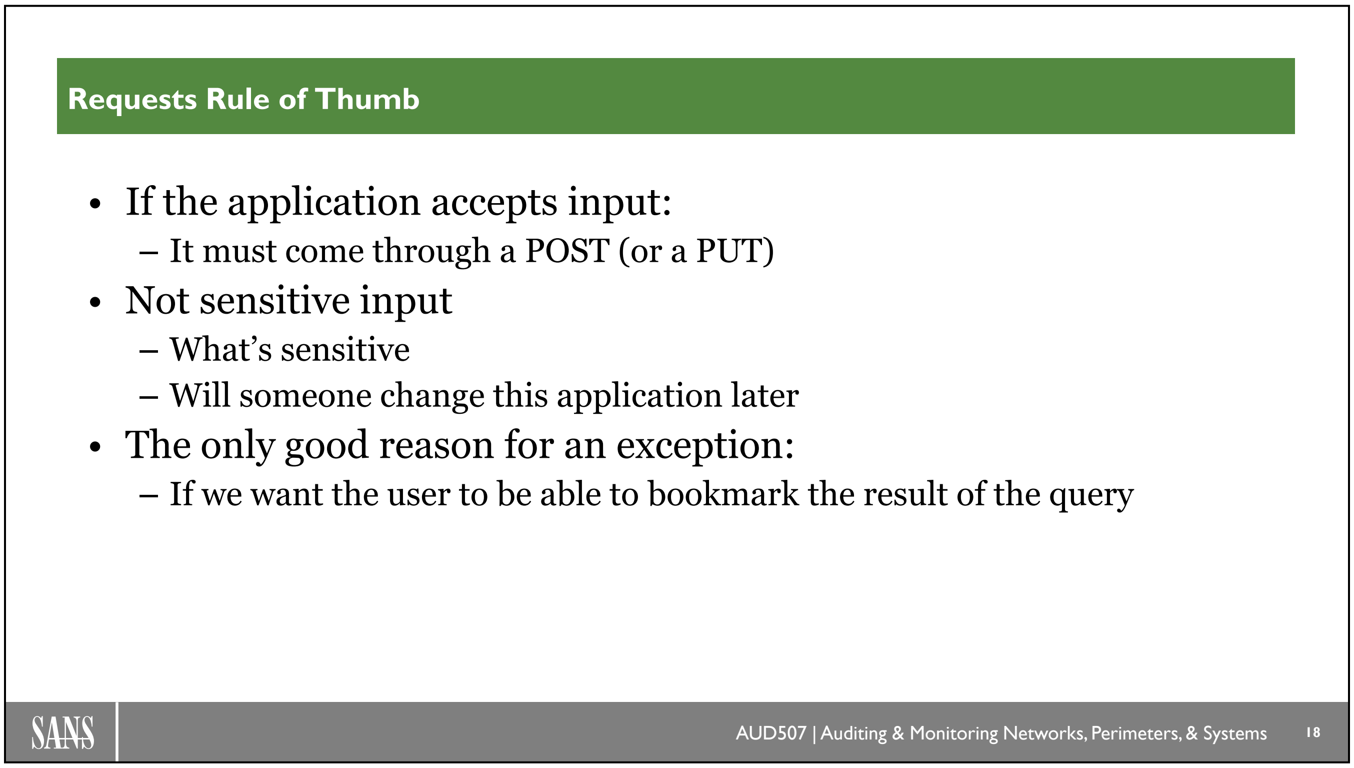
Another important header to notice in this example is the Cookie header. Cookies will be discussed more deeply later in this material.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Requests Rule of Thumb

- If the application accepts input:
  - It must come through a POST (or a PUT)
- Not sensitive input
  - What's sensitive
  - Will someone change this application later
- The only good reason for an exception:
  - If we want the user to be able to bookmark the result of the query

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 18

Before we go on, let's take this opportunity to spell out a rule of thumb that should be incorporated into your web application design practices. If you have an application that is going to accept input, that input must be submitted through a POST rather than a GET. Technically, it could also be sent over a PUT, which we can think of as an alternative kind of POST, but not many applications today use this request method.

This is not to say that it is not technically possible to submit using a GET. Clearly, it is. Rather, we are saying that best practice is that if you are accepting input from a form, require that it be submitted via a POST request.

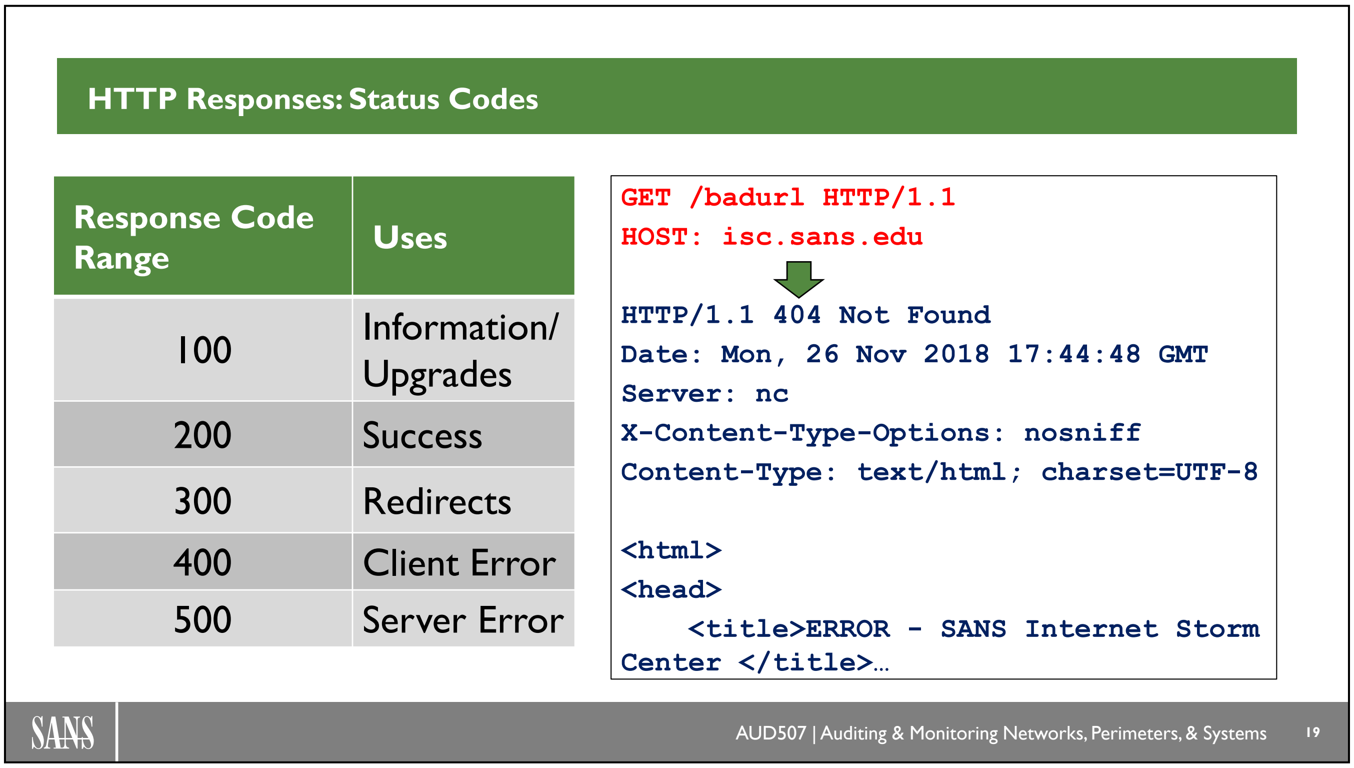
Some developers may argue that the form in question doesn't actually contain any sensitive data. There are some holes in this position. First, they are simply defending a bad practice that they have already followed, and they don't want to rewrite code. Second, it is no more difficult to handle input coming from a GET or a POST; in fact, from the programmer's point of view, it is often handled identically through the web application programming API. Lastly, and perhaps most important, although there may not be anything on that form that is sensitive *today*, who is to say how that form will change over time? For this reason, we recommend this as a coding requirement.

The only good reason to violate this rule of thumb would be in a situation where you would like to permit the user to bookmark the result of the query. A perfect example of this is the use of a search engine. Imagine that a user comes to our site, browses around, and ultimately searches to find some piece of documentation. Rather than sending his friend the document, he instead copies the current URL and sends that to his friend, allowing him to see how to search for that same item himself.

The bottom line is that using a GET to submit a form should be the exception. Using a POST to submit a form should be the rule.

August 10, 2021

© SANS Institute 2021



HTTP Responses: Status Codes

Response Code Range	Uses
100	Information/ Upgrades
200	Success
300	Redirects
400	Client Error
500	Server Error

```
GET /badurl HTTP/1.1
HOST: isc.sans.edu

HTTP/1.1 404 Not Found
Date: Mon, 26 Nov 2018 17:44:48 GMT
Server: nc
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=UTF-8

<html>
<head>
   <title>ERROR - SANS Internet Storm
Center </title>…
```

When a web server sends a request to a client, it will include a "response status code," which represents the results of the request which the server just processed. These status codes are grouped into five categories.

100's: The 100-series codes are used for informational messages and to communicate information about protocol upgrades which have been requested by the client. For example, if a web browser requests that the server upgrade the connection to use either HTTP/2 or Web Sockets, the server may respond with a "101 Switching Protocols" status code, notifying the client that it is okay to switch protocols.

200's: These status codes represent that the request was processed successfully. The most commonly seen code is "200 OK," which means that the message body contains the appropriate results for the request.

300's: The 300-series messages are used to redirect the client to another resource. Normally, the browser will follow up on a 300-series message by requesting the new resource given in the response. A common message is "301 Moved Permanently," which instructs the browser always to use the new URL to access the requested resource.

400's: These messages are used to tell the web client that its request could not be successfully processed because it was somehow made in error. The "404 Not Found" message tells the browser that it has requested a resource that does not exist, for instance.

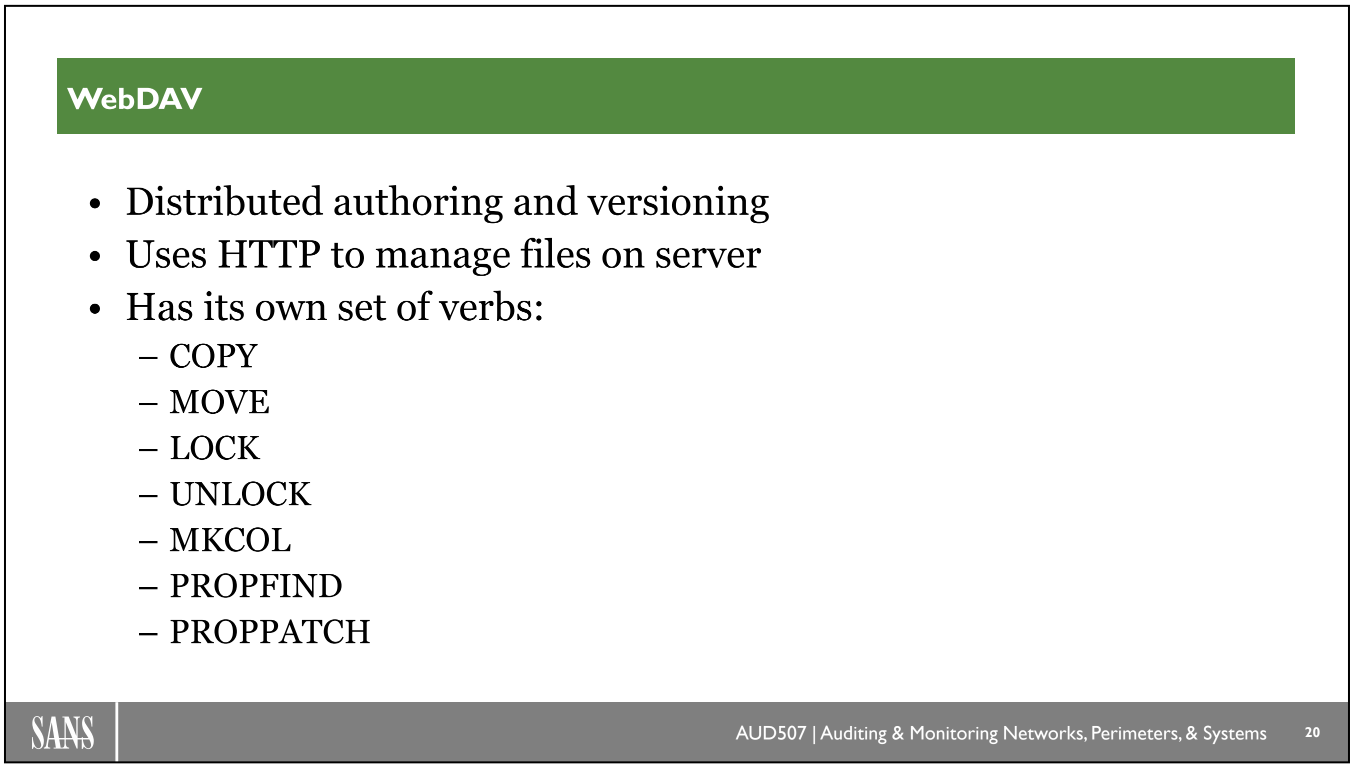
500's: The 500-series messages indicate that an error occurred on the server during processing of the request. A common error in this series is the "500 Internal Server Error," which tells the client that the server experienced an unexpected error, from which it could not recover.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



WebDAV

- Distributed authoring and versioning
- Uses HTTP to manage files on server
- Has its own set of verbs:
  - COPY
  - MOVE
  - LOCK
  - UNLOCK
  - MKCOL
  - PROPFIND
  - PROPPATCH

Web distributed authoring and versioning (WebDAV) was designed to allow web developers to manage content on a web server using clients which speak HTTP. WebDAV has a set of HTTP verbs all its own, which are geared toward managing the resources on a server.

COPY is used to copy a resource to a new location.

MOVE is used to move a resource to a new location.

LOCK allows the client to place either a shared or exclusive lock on a resource before changing it.

UNLOCK removes a resource lock created with a LOCK operation.

MKCOL is used to create a new collection (usually a directory) of resources on the server.

PROPFIND may be used to get the properties of a resource or to retrieve a directory listing from the server.

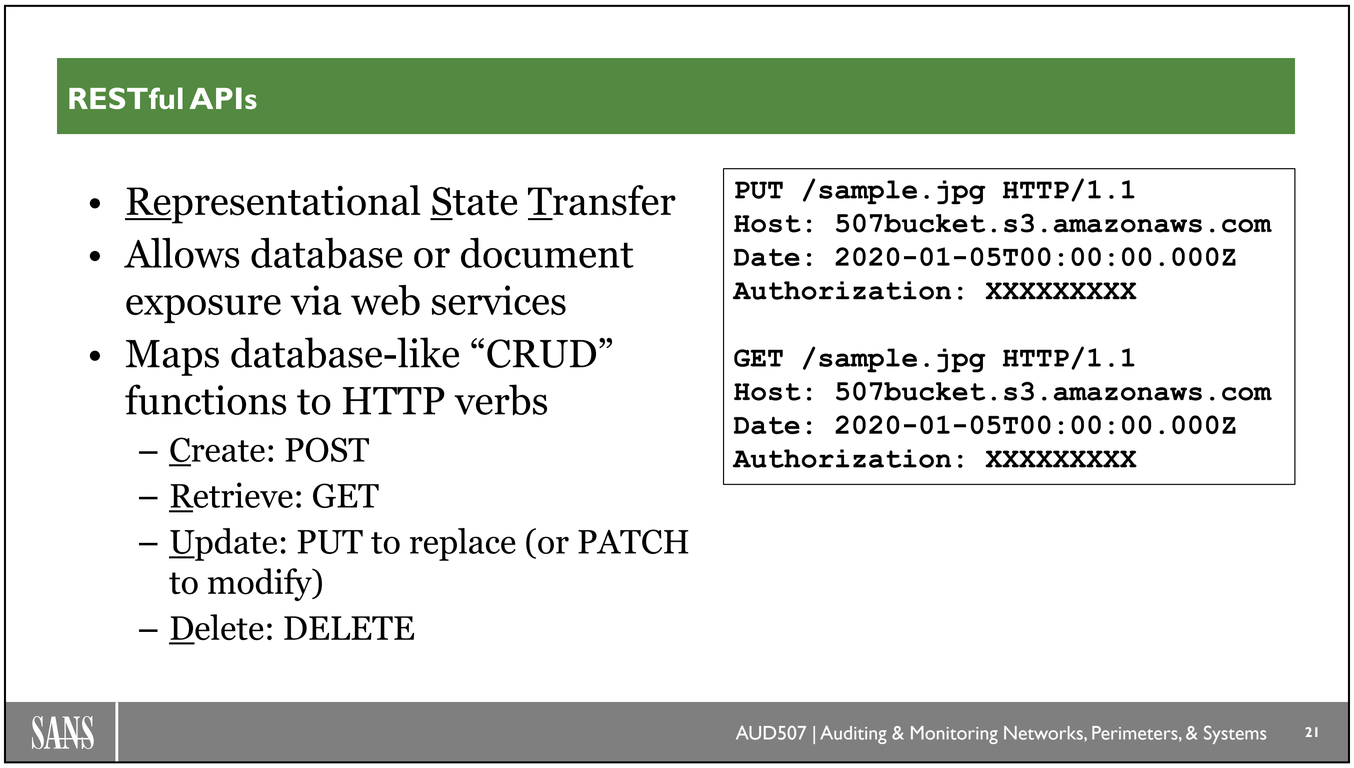
PROPPATCH allows for editing of properties on a resource.

Our method of testing applications by analyzing and manipulating HTTP requests and responses will work with WebDAV as well as with any other HTTP-based applications and protocols.

August 10, 2021

August 10, 2021

© SANS Institute 2021



RESTful APIs

- Representational State Transfer
- Allows database or document exposure via web services
- Maps database-like "CRUD" functions to HTTP verbs
  - Create: POST
  - Retrieve: GET
  - Update: PUT to replace (or PATCH to modify)
  - Delete: DELETE

```
PUT /sample.jpg HTTP/1.1
Host: 507bucket.s3.amazonaws.com
Date: 2020-01-05T00:00:00.000Z
Authorization: XXXXXXXXX

GET /sample.jpg HTTP/1.1
Host: 507bucket.s3.amazonaws.com
Date: 2020-01-05T00:00:00.000Z
Authorization: XXXXXXXXX
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 21

RESTful services were introduced in 2000 by Roy Fielding as part of his doctoral dissertation at the University of California, Irvine.

REST APIs – which are not always in full compliance with Fielding's design – are used by many cloud services to provide management of database entities or documents within their services. These services use HTTP verbs to represent the create, retrieve, update, and delete functions that a client can perform on managed objects.

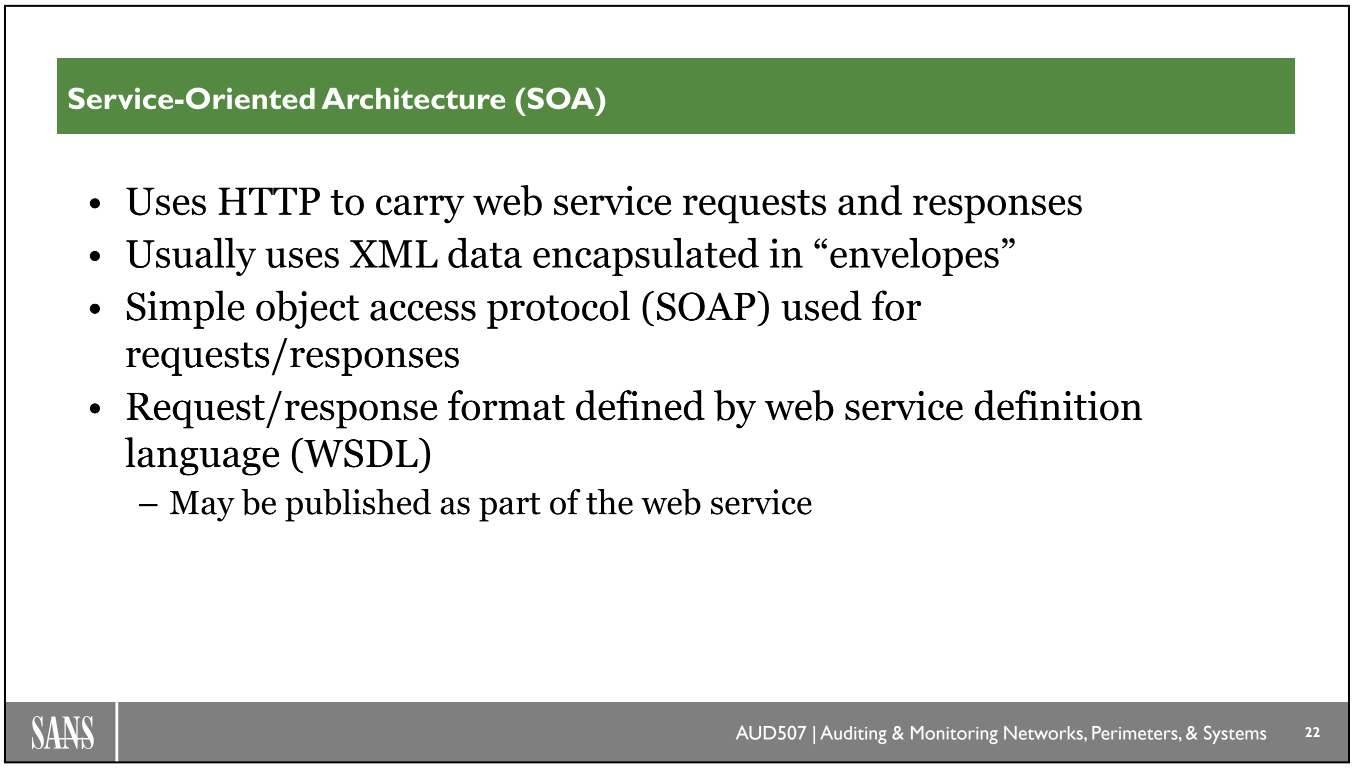
A good example of a RESTful API is the Amazon Simple Storage Service (S3) API, which is well documented at the AWS website.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Service-Oriented Architecture (SOA)

- Uses HTTP to carry web service requests and responses
- Usually uses XML data encapsulated in “envelopes”
- Simple object access protocol (SOAP) used for requests/responses
- Request/response format defined by web service definition language (WSDL)
  - May be published as part of the web service

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 22

In service-oriented architecture (SOA), web services exchange XML-formatted data with clients, using interfaces defined in a web services definition language (WSDL) document. SOA services use HTTP to carry requests and responses which have been defined in the WSDL. The data transferred is in a special XML (extensible markup language) format known as an “envelope.” These XML requests implement the simple object access protocol, or SOAP.

The WSDL for a web service can be published as a resource in the HTTP directory structure of the service, i.e., https://svc.domain.com/svc/svc.wsdl for a (fictitious) public web service or could be stored offline for private services. A portion of an example WSDL found on the internet is shown on the next page.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

SOAP Example: WSDL for Calculator Service

```
<wsdl:definitions ...>
  <wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
  <s:element name="Add">
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
        <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
      </s:sequence>
    </s:complexType>
  </s:element>
  ...
</wsdl:definitions>
```

This excerpt from a SOA WSDL file shows a template for the “Add” function of a calculator service. The template includes variables called “IntA” and “IntB,” which are the operands for the addition function. A client using this service will use the WSDL definitions to create a properly-formatted request to the service to use the add operation. Other operations, like subtract, etc. are included in the same WSDL file.

Excerpts from XML envelopes containing a request and response to this service are shown on the following pages.

These examples were taken from a sample web service online.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

SOAP Request

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns:tem="http://tempuri.org/">
   <soap:Header/>
   <soap:Body>
      <tem:Add>
         <tem:intA>1</tem:intA>
         <tem:intB>2</tem:intB>
      </tem:Add>
   </soap:Body>
</soap:Envelope>
```

In this SOAP request, the client is asking the web service to perform the “add” operation on the integer values 1 and 2, stored in the variables IntA and IntB.

The request is created in XML using the document structure of a SOAP envelope. The envelope contains the SOAP header and all the data required to be passed as part of the request.

The next page shows the SOAP response from the web service.

August 10, 2021

August 10, 2021

© SANS Institute 2021

SOAP Response

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <AddResponse xmlns="http://tempuri.org/">
         <AddResult>3</AddResult>
      </AddResponse>
   </soap:Body>
</soap:Envelope>
```

This page shows an excerpt from the web service's response to the previous request. In an XML document, the service returns an "AddResponse," which contains an "AddResult" of 3, the result of adding our two integers together.

August 10, 2021

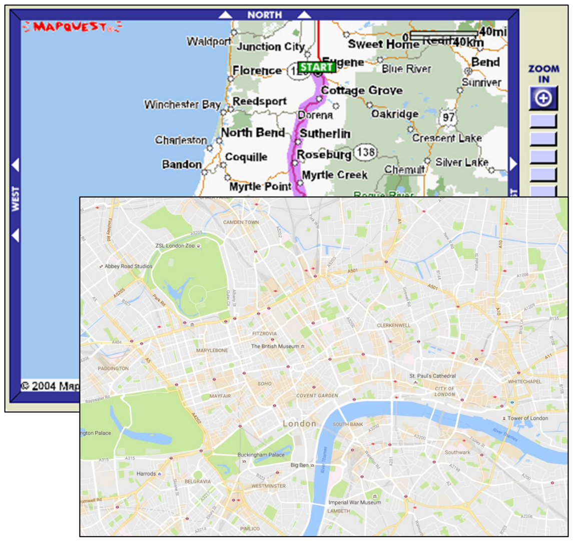
August 10, 2021

Technet24

© SANS Institute 2021

AJAX

- Asynchronous JavaScript and XML:
  - Google Maps is a perfect example
  - Page loads using HTTP
  - JavaScript asynchronously requests more data
  - Page updates without reloading
- #1 AJAX Security Problem:
  - Bypassing authentication or authorization system with "bolt on" AJAX



Another important technology to keep your eye on is AJAX. AJAX is still a relatively new technique, though it is based on technologies that have been around for quite some time.

Essentially, AJAX enables you to use JavaScript to asynchronously (or, while you're doing something else) request data (possibly sent using XML) from the server and update the page without having the entire page refresh. Right now, this is a hot technology, and we see it evolving in several forms. Although AJAX is the formal name that these are known by, not all the solutions necessarily use JavaScript or XML, but they perform the same type of function.

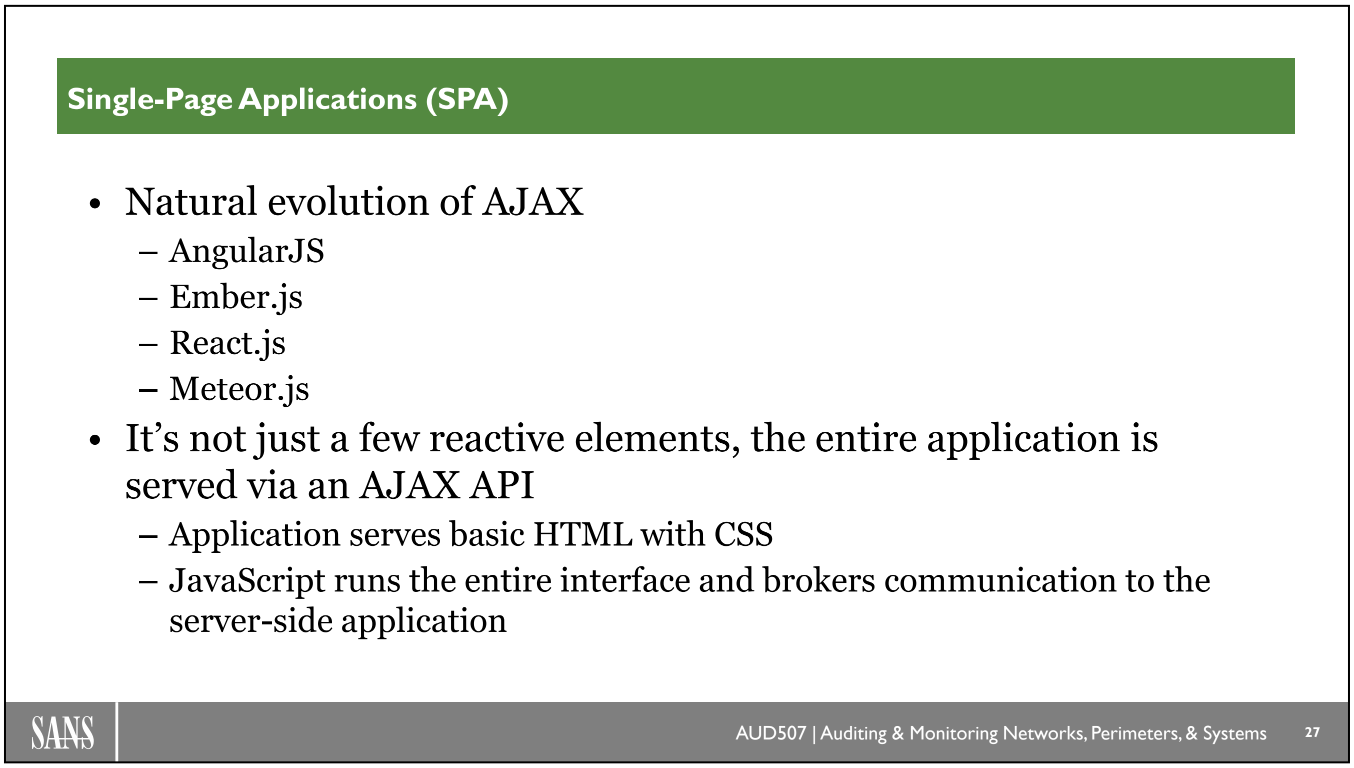
A great example of this technology in action is Google Maps. When you use the Google Maps site, the page quickly loads and displays a small piece of the map. In the background (asynchronously), the page then begins to cache neighboring pieces of map. Similarly, if you zoom in, the client-side application gives you a rough zoom while the background process requests and loads the details, filling them in as they come. Contrast this with the old interface for MapQuest, which had arrows on each side of the map. When the user clicked an arrow, the page would reload with the map shifted in that direction. Each click caused a distinct refresh of the page.

Although you have likely heard of a large number of vulnerabilities involving AJAX in recent times, there is nothing actually inherently insecure about AJAX. Problems arise when a programmer is tasked to or decides to add some AJAX to an already existing application and fails to correctly integrate it with the security system.

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Single-Page Applications (SPA)

- Natural evolution of AJAX
  - AngularJS
  - Ember.js
  - React.js
  - Meteor.js
- It's not just a few reactive elements, the entire application is served via an AJAX API
  - Application serves basic HTML with CSS
  - JavaScript runs the entire interface and brokers communication to the server-side application

A natural evolution of the use of AJAX to make pages more responsive is the development of the entire frontend UI of a web application in JavaScript. The name for this is, "single-page application." By taking this approach, it is possible to make the web application appear to be, effectively, a desktop application. The entire interface is dynamically built using the Document Object Model (DOM) and JavaScript coupled with Cascading Style Sheets (CSS).

When this approach is used in the extreme, there may be very little actual HTML transmitted as the content of a webpage. Instead, successive requests are made over AJAX via an exposed API on the server side in reaction to the user. The results of these requests could be more JavaScript code that is executed to create some effect, HTML that will be rendered onto the browser canvas by the JavaScript interface, data that will be used to populate dynamically generated tables, and more.

There are a growing number of frameworks that are used to create these highly responsive web applications. Some of the best known include AngularJS, Ember.js, React.js, and Meteor.js. While these have become very popular for rapid responsive web application development, you should be aware that developers make a number of very common errors with these frameworks. The very first is that they forget that the entirety of the source code, the JavaScript, for the interface is delivered to the user in the web browser. In most of the applications that we have examined, the interface code delivered will include *all* of the features and functions that the application contains, even if the user does not currently have access to those features. Along with this information, the mechanisms used to make those requests through the server-side API are also present. This makes it trivial to begin attacking the application. Since these APIs are often weakly protected or even fail to implement authorization checks completely, it becomes possible to compromise the confidentiality or integrity of the site.

August 10, 2021

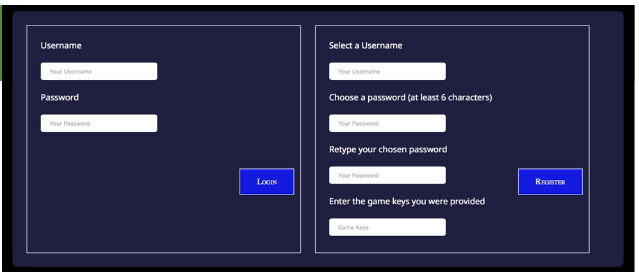
August 10, 2021

Technet24

© SANS Institute 2021

Source of a Typical SPA

- Where is the <form> tag



```
<!doctype html><html lang="en"><head><meta charset="utf-8"><title>SEC 503 Score
Server</title><base href="/"><meta name="viewport" content="width=device-
width,initial-scale=1"><link rel="icon" type="image/x-icon"
ngHref="favicon.ico"><link href="styles.441e32aac05d77dab7aa.bundle.css"
rel="stylesheet"/></head><body><app-root></app-root><script
type="text/javascript"
src="inline.10ae9b5b5bc5783383b7.bundle.js"></script><script
type="text/javascript"
src="polyfills.b89f6da3f94d82c339a7.bundle.js"></script><script
type="text/javascript"
src="main.75f0bb7ffc3642fb60d8.bundle.js"></script></body></html>
```

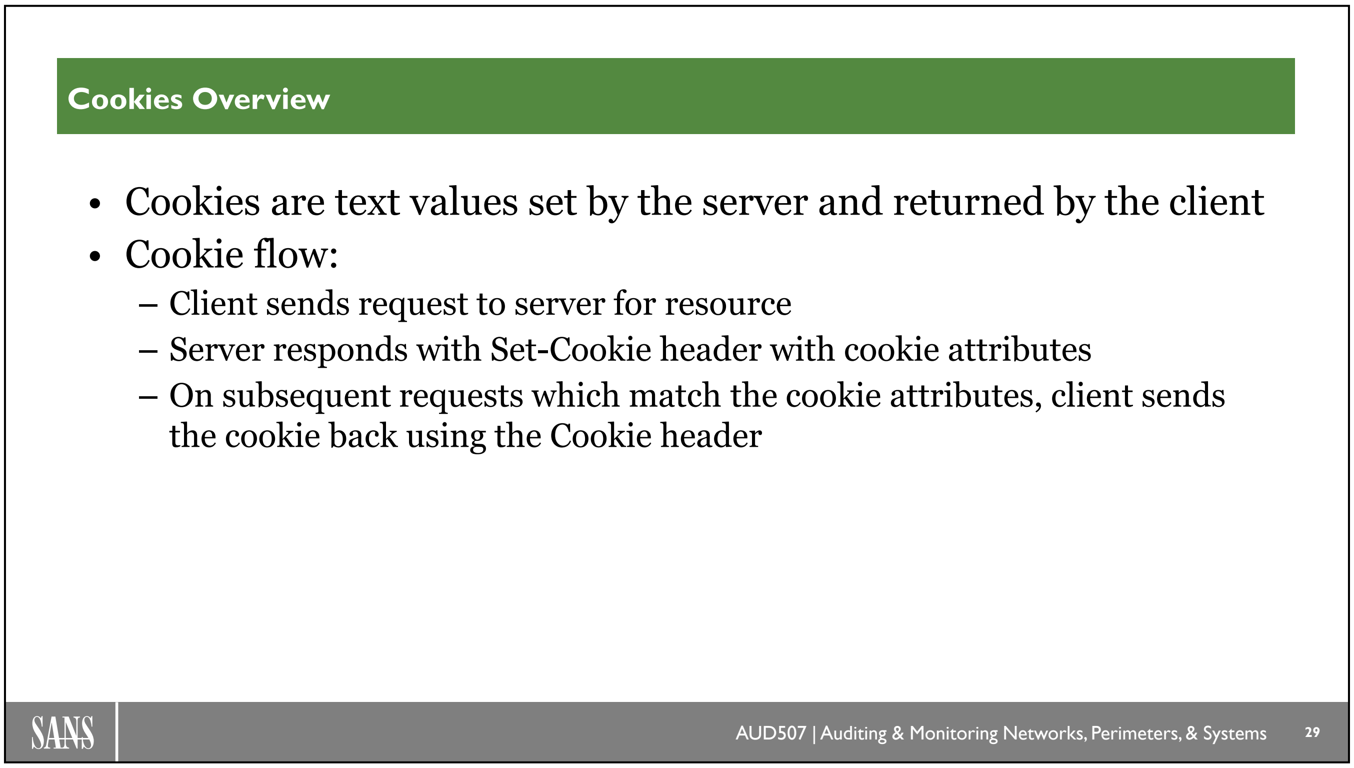
Consider, for example, the content in the slide above. This is the typical source code for a single-page application these days. You'll notice that the source code displayed bears absolutely no resemblance to the webpage in the upper right corner. In fact, every single element being displayed on that webpage is being generated by a JavaScript application that shows up as the last item in the source code.

There are certainly JavaScript de-obfuscators that can be used to perform code-level analysis of an application such as this, but there truly is no need to do so. If we simply use our browser to send our requests through a proxy, like Burp, we have an opportunity to see how the application behaves.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Cookies Overview

- Cookies are text values set by the server and returned by the client
- Cookie flow:
  - Client sends request to server for resource
  - Server responds with Set-Cookie header with cookie attributes
  - On subsequent requests which match the cookie attributes, client sends the cookie back using the Cookie header

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 29

We mentioned earlier that HTTP is stateless – a server will not remember that a client has ever visited the site before. This makes it difficult to keep up with things like who is using the site, what items they have in their shopping cart, etc.

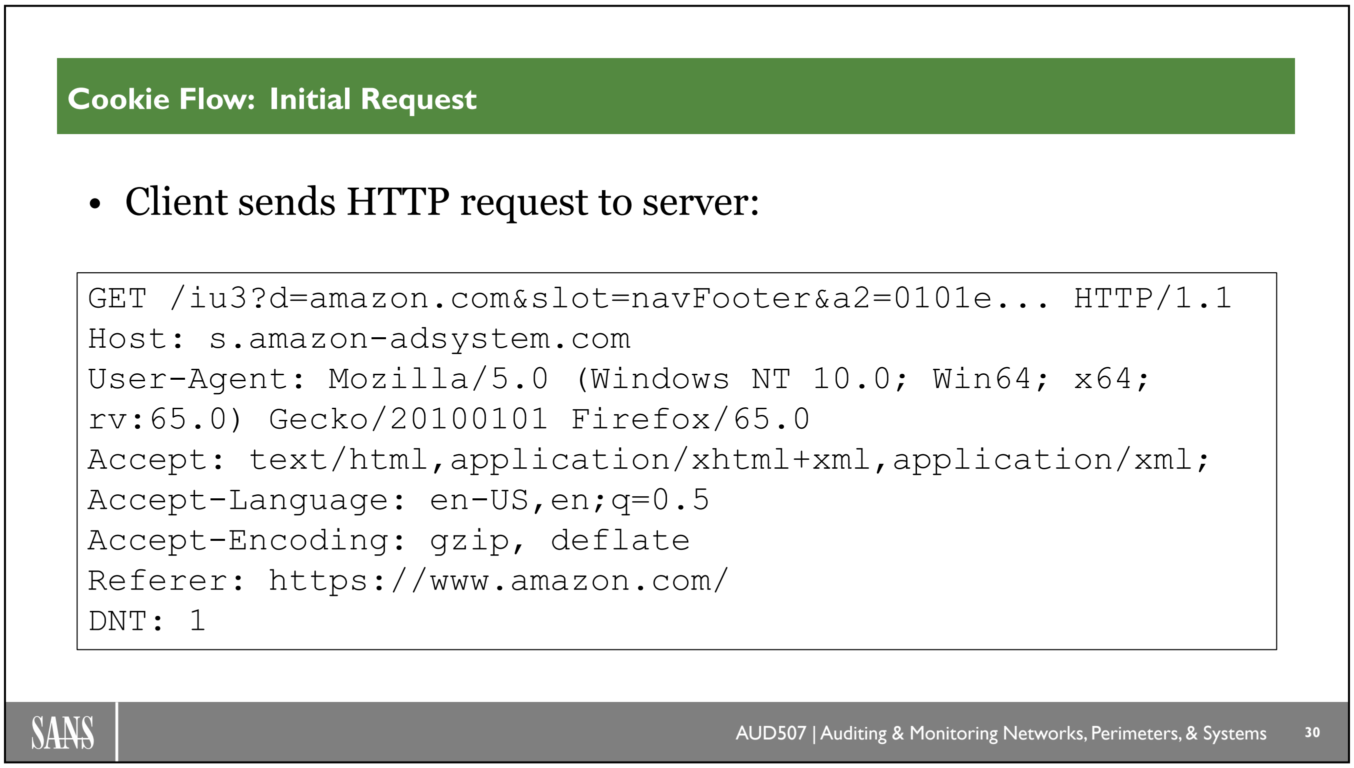
Cookies are one mechanism that can be used to add some statefulness to our web applications. Cookies are bits of text data that are set by the server, and then returned by the web browser on subsequent visits. The data is sent from the server in a header called “Set-Cookie” and returned by the browser in a header simply called “Cookie.” The cookie is only returned by the browser if all of the cookie’s attributes allow it to be sent. We’ll cover those attributes and their meanings on the next several slides.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Cookie Flow: Initial Request

- Client sends HTTP request to server:

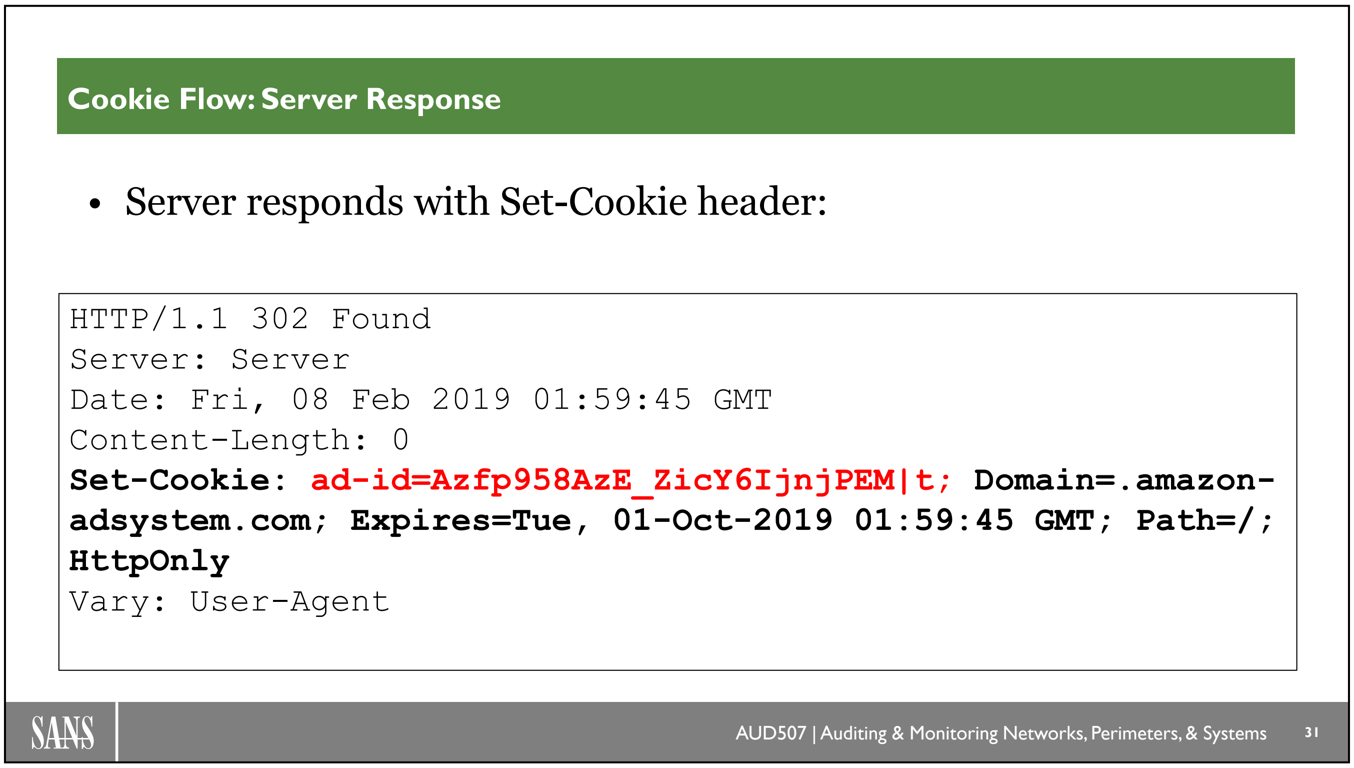
```
GET /iu3?d=amazon.com&slot=navFooter&a2=0101e... HTTP/1.1
Host: s.amazon-adsystem.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.amazon.com/
DNT: 1
```

The next few slides present an example of a cookie being set by an Amazon web application. In this slide, the browser has sent its initial request for some resource on the site. In later traffic, the server will set a cookie and the browser will return that cookie to the server in all appropriate requests (those where all of the cookie attributes match).

August 10, 2021

August 10, 2021

© SANS Institute 2021



Cookie Flow: Server Response

- Server responds with Set-Cookie header:

```
HTTP/1.1 302 Found
Server: Server
Date: Fri, 08 Feb 2019 01:59:45 GMT
Content-Length: 0
Set-Cookie: ad-id=Azfp958AzE_ZicY6IjnjPEM|t; Domain=.amazon-
adsystem.com; Expires=Tue, 01-Oct-2019 01:59:45 GMT; Path=/;
HttpOnly
Vary: User-Agent
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 31

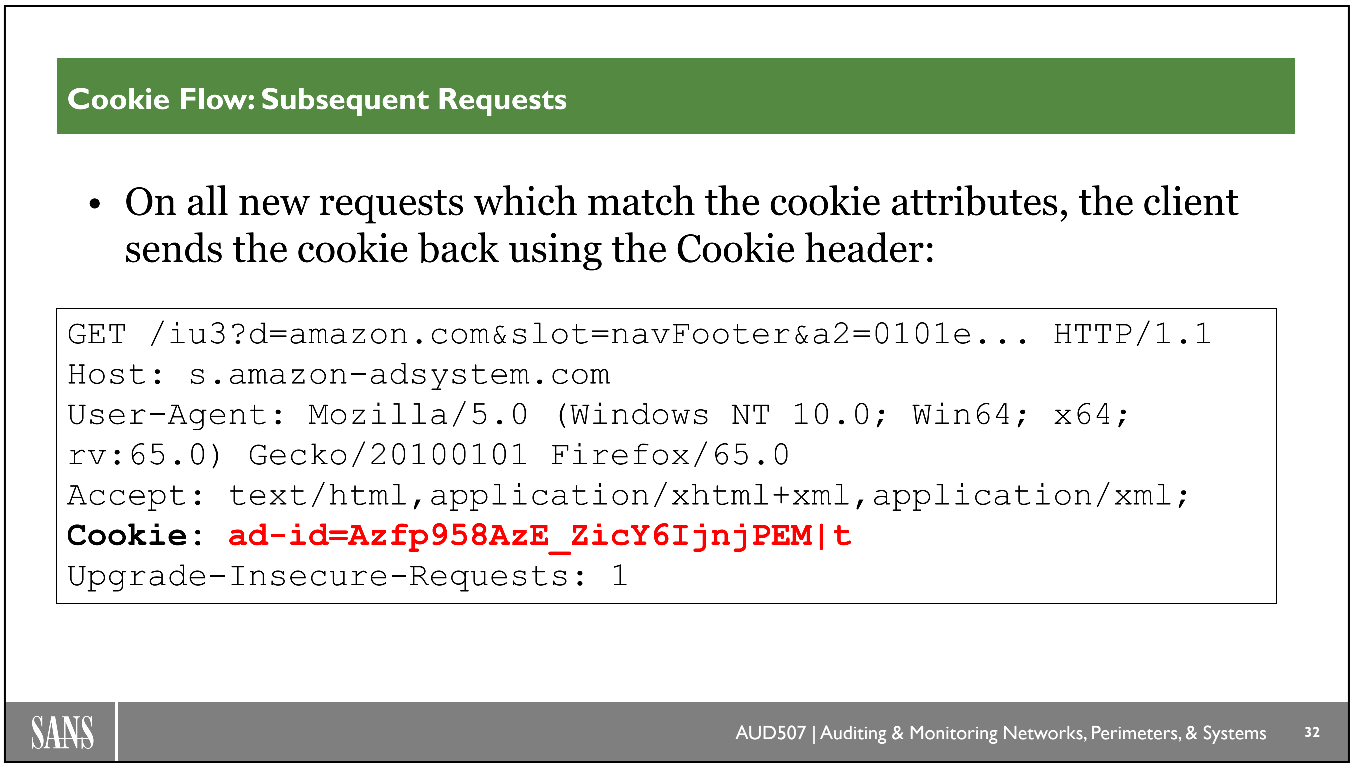
In this HTTP response, the server uses the "Set-Cookie" header to send a cookie to the browser. Remember that every cookie has at least a name=value pair. In this cookie the name is "ad-id," and the value is "Azfp958AzE_ZicY6IjnjPEM|t." The semicolon is used to separate multiple name=value pairs in the same header. The other parts of the cookie, like "Domain," "Expires," "Path," and "HTTPOnly" are the attributes of this cookie. These will be covered on their own slides.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Cookie Flow: Subsequent Requests

- On all new requests which match the cookie attributes, the client sends the cookie back using the Cookie header:

```
GET /iu3?d=amazon.com&slot=navFooter&a2=0101e... HTTP/1.1
Host: s.amazon-adsystem.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;
Cookie: ad-id=Azfp958AzE_ZicY6IjnjPEM|t
Upgrade-Insecure-Requests: 1
```

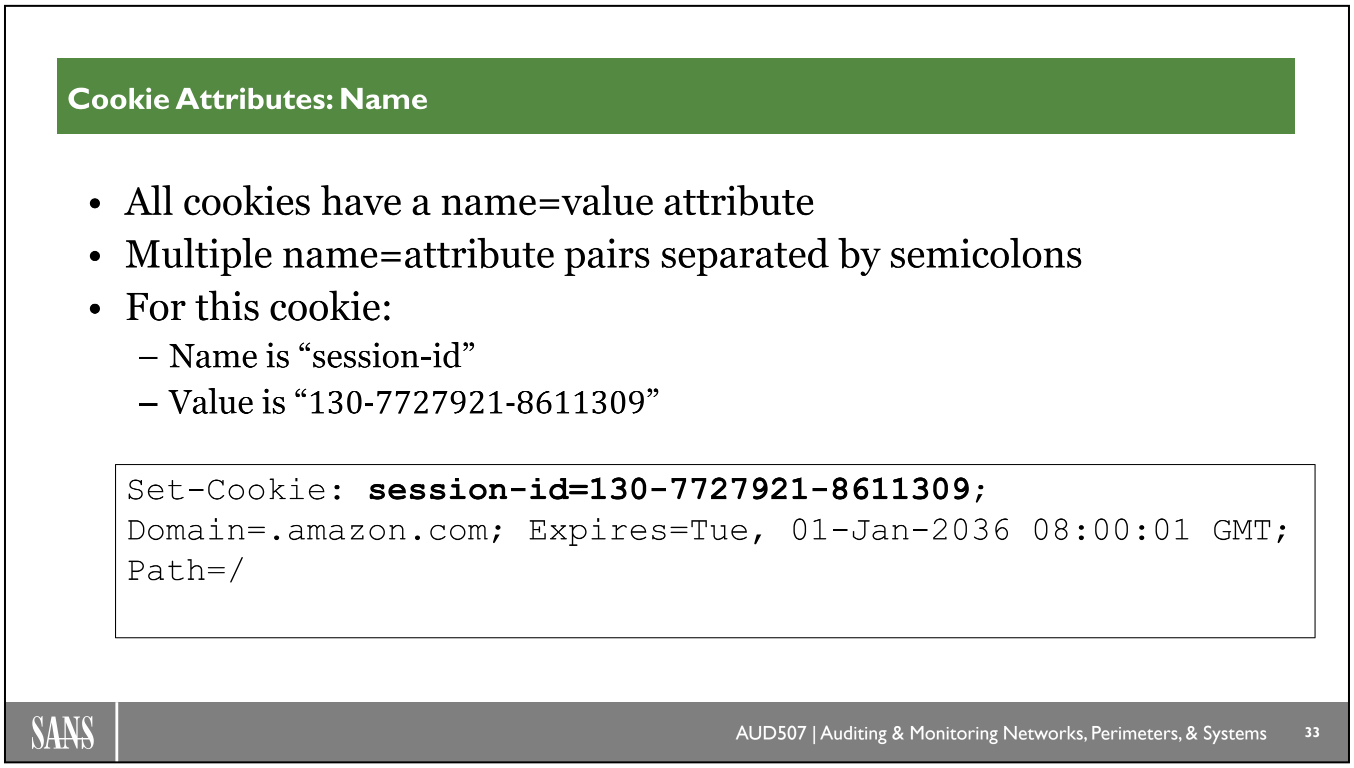
Now that the browser has a copy of the cookie, it will send the cookie value back to the server on every subsequent request (as long as all of the attributes correctly match the request).

In this request, the browser is sending back the cookie we saw set on the previous slide by using the “Cookie:” header with the name=value pair from the cookie.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Cookie Attributes: Name

- All cookies have a name=value attribute
- Multiple name=attribute pairs separated by semicolons
- For this cookie:
  - Name is “session-id”
  - Value is “130-7727921-8611309”

```
Set-Cookie: session-id=130-7727921-8611309;
Domain=.amazon.com; Expires=Tue, 01-Jan-2036 08:00:01 GMT;
Path=/
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 33

Here is another example of the name=value pair for a cookie. In this example, the name is “session-id” and the associated value is “130-7727921-8611309.”

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Cookie Attributes: Domain and Path

- The Domain attribute instructs the browser to send the cookie only to the specified domain
- The Path attribute specifies the beginning of the path which *must* be included in the URL to return the cookie
- NOTE: Cookie attribute names are not case-sensitive

```
Set-Cookie: ad-id=Azfp958AzE_ZicY6IjnjPEM|t;
Domain=.amazon-adsystem.com; Expires=Tue, 01-Oct-2019
01:59:45 GMT; Path=/; HttpOnly
```

Two commonly used cookie attributes are “Domain” and “Path.”

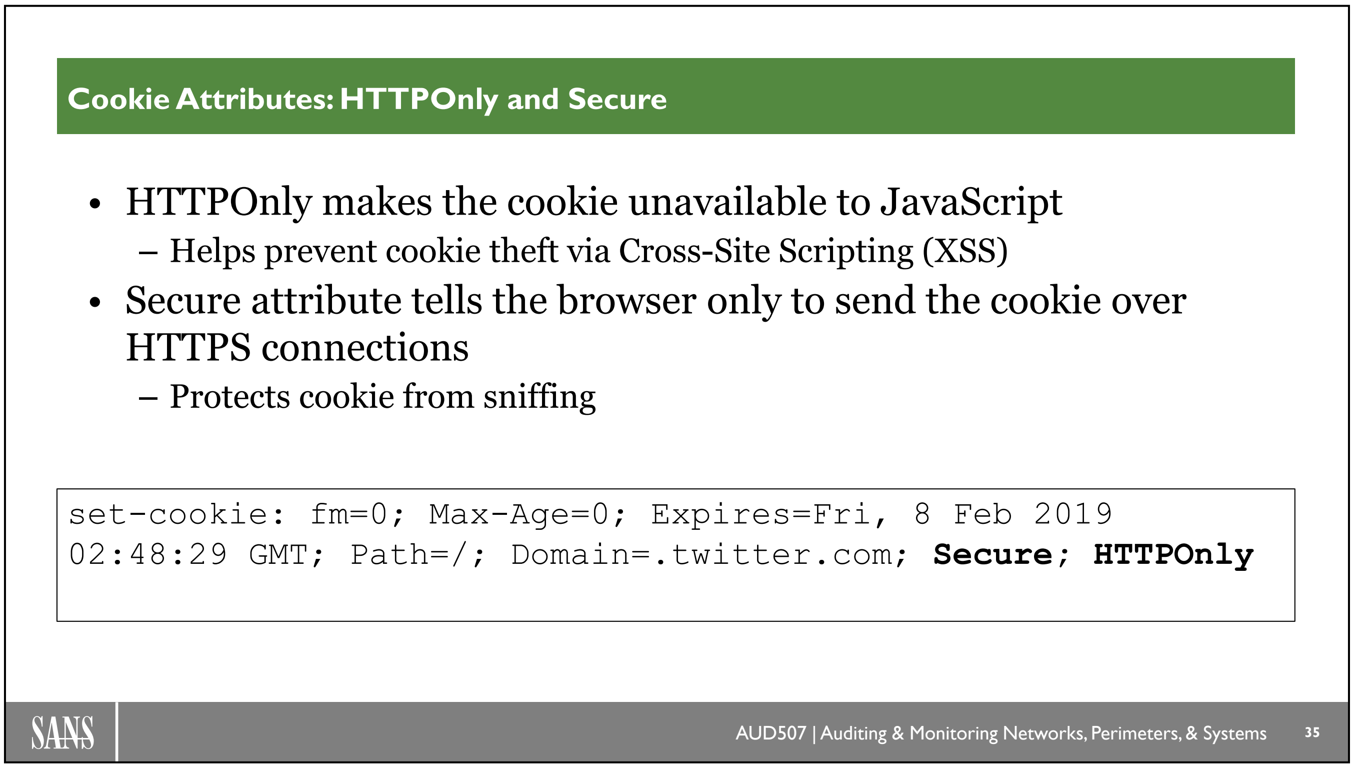
The “Domain” attribute instructs the browser that it should only send this cookie back during requests for resources from a domain that matches. In this example, the cookie *would* be sent to www.amazon-adsystem.com because it matches the attribute value. This cookie *would not* be sent to www.amazon.com, because that does not match the attribute value.

The “Path” attribute tells the browser to return this cookie only if the path portion of the URL (the part that comes after the hostname) *begins with* the path set in the cookie. Because this cookie has a “Path” attribute of “/,” every path on the server will match. Therefore, this cookie will be sent back for requests for ANY path on this server.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Cookie Attributes: HTTPOnly and Secure

- HTTPOnly makes the cookie unavailable to JavaScript
  - Helps prevent cookie theft via Cross-Site Scripting (XSS)
- Secure attribute tells the browser only to send the cookie over HTTPS connections
  - Protects cookie from sniffing

```
set-cookie: fm=0; Max-Age=0; Expires=Fri, 8 Feb 2019
02:48:29 GMT; Path=/; Domain=.twitter.com; Secure; HTTPOnly
```

SANS

AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 35

When we discuss session handling later, we'll discover that cookies are frequently used to contain session identifiers for users. These session identifiers can be used to prove that the user is authenticated and can access restricted parts of the web application. Attackers frequently try to steal such "session cookies," so that they can impersonate an authenticated user on a website. One commonly used method of cookie theft is Cross-Site Scripting (XSS), in which the attacker tries to force JavaScript code into the user's browser to grab a copy of the session cookie and send it to the attacker.

The "HTTPOnly" flag serves to mitigate this attack by instructing the browser to make this cookie unavailable for access by JavaScript. The attacker's script can't steal what it can't see.

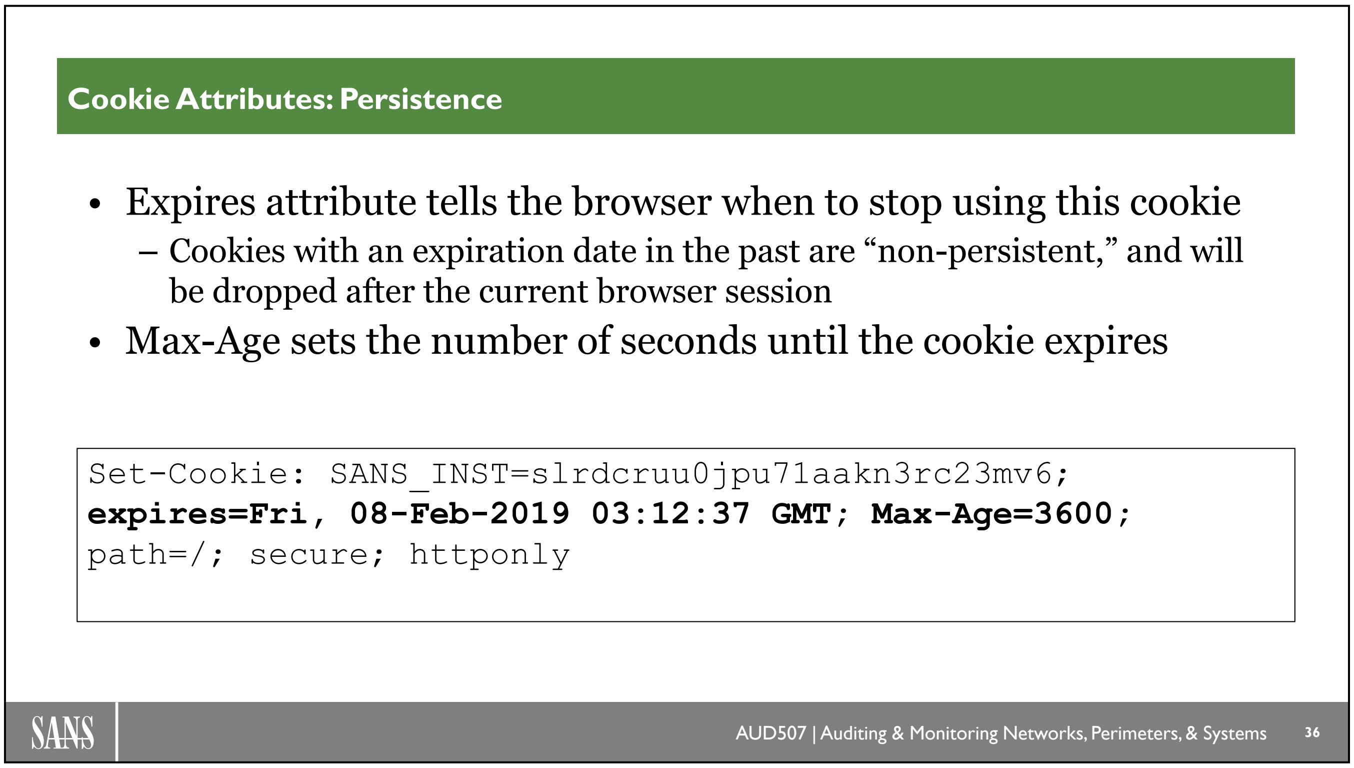
The "Secure" attribute instructs the browser to return this cookie only over encrypted HTTPS connections – never over plaintext HTTP. This helps to protect the cookie against being intercepted, say by an attacker sniffing wireless traffic in a coffee shop.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Cookie Attributes: Persistence

- Expires attribute tells the browser when to stop using this cookie
  - Cookies with an expiration date in the past are "non-persistent," and will be dropped after the current browser session
- Max-Age sets the number of seconds until the cookie expires

```
Set-Cookie: SANS_INST=slrdcruu0jpu71aakn3rc23mv6;
expires=Fri, 08-Feb-2019 03:12:37 GMT; Max-Age=3600;
path=/; secure; httponly
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 36

The "Expires" attribute is the older way of letting a browser know to stop using a cookie after a certain date and time. If the developer wishes to make a cookie non-persistent, she can set the expiration date in the past; the browser will use the cookie for *this* session but will discard the cookie when the browser is closed.

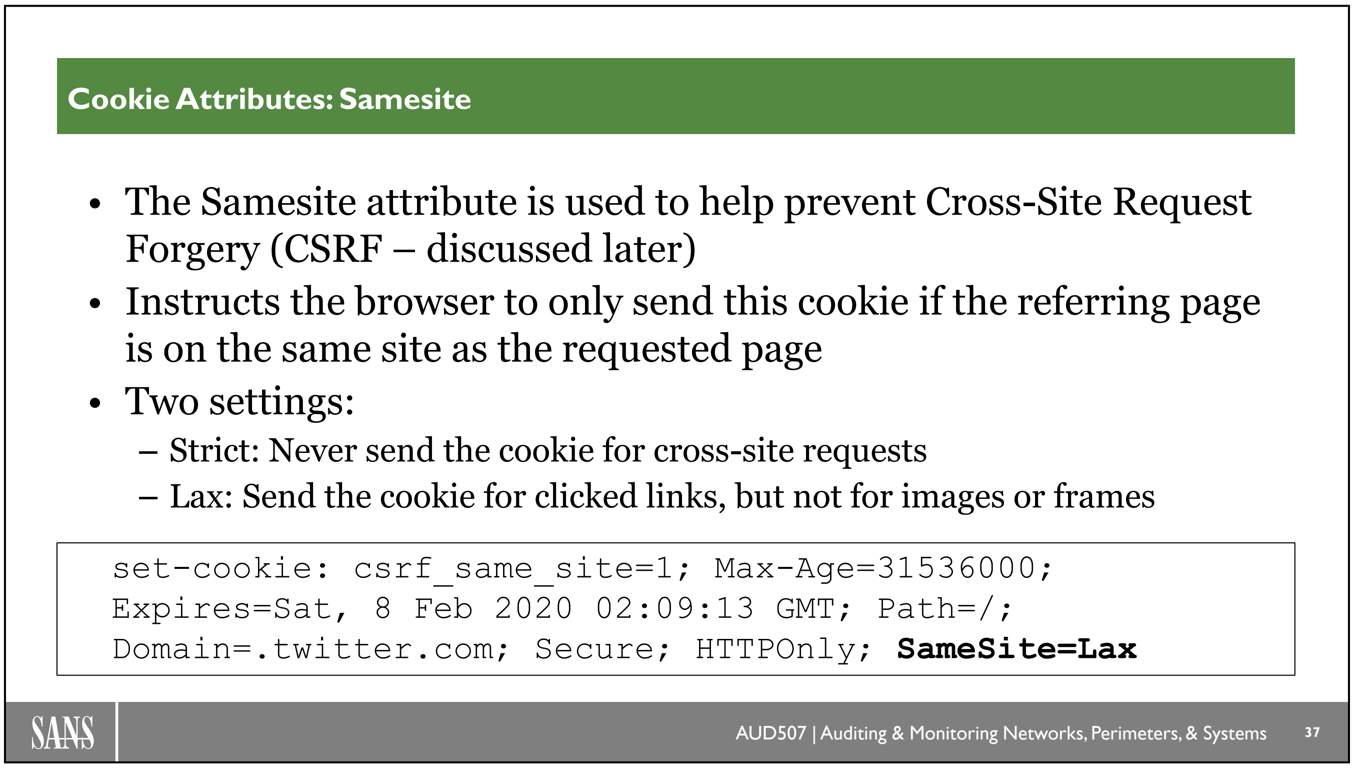
"Max-Age" is the newer way of setting expiration for cookies. It simply specifies the number of seconds the cookie should be kept after it is first received. Setting this value to zero will make the cookie non-persistent.

Many developers will set both cookie flags since some older browsers do not honor the "Max-Age" flag.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Cookie Attributes: Samesite

- The Samesite attribute is used to help prevent Cross-Site Request Forgery (CSRF – discussed later)
- Instructs the browser to only send this cookie if the referring page is on the same site as the requested page
- Two settings:
  – Strict: Never send the cookie for cross-site requests
  – Lax: Send the cookie for clicked links, but not for images or frames

```
set-cookie: csrf_same_site=1; Max-Age=31536000;
Expires=Sat, 8 Feb 2020 02:09:13 GMT; Path=/;
Domain=.twitter.com; Secure; HTTPOnly; SameSite=Lax
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 37

The "Samesite" attribute instructs the browser to return the cookie only when the page that originated the request comes from the "same site." The concept of "same site" is understood by most browsers to mean a URL with the same protocol, hostname, and port number. Some versions of Internet Explorer are an exception to this rule, as they do not consider the port number and may ignore the same site restrictions for sites in "Trusted Zones."

This attribute is used to mitigate the potential damage from an attack (discussed later) know as Cross-Site Request Forgery (CSRF). This attack is used to trick a browser into acting on the user's behalf, without the user knowing about it. We'll discuss CSRF later in the section, but for now, it's enough to know that this attribute will limit the browser's ability to send cookies to a site during such cross-site requests.

There are two modes associated with this attribute:

In strict mode, the browser will never send the cookie back for cross-site requests.

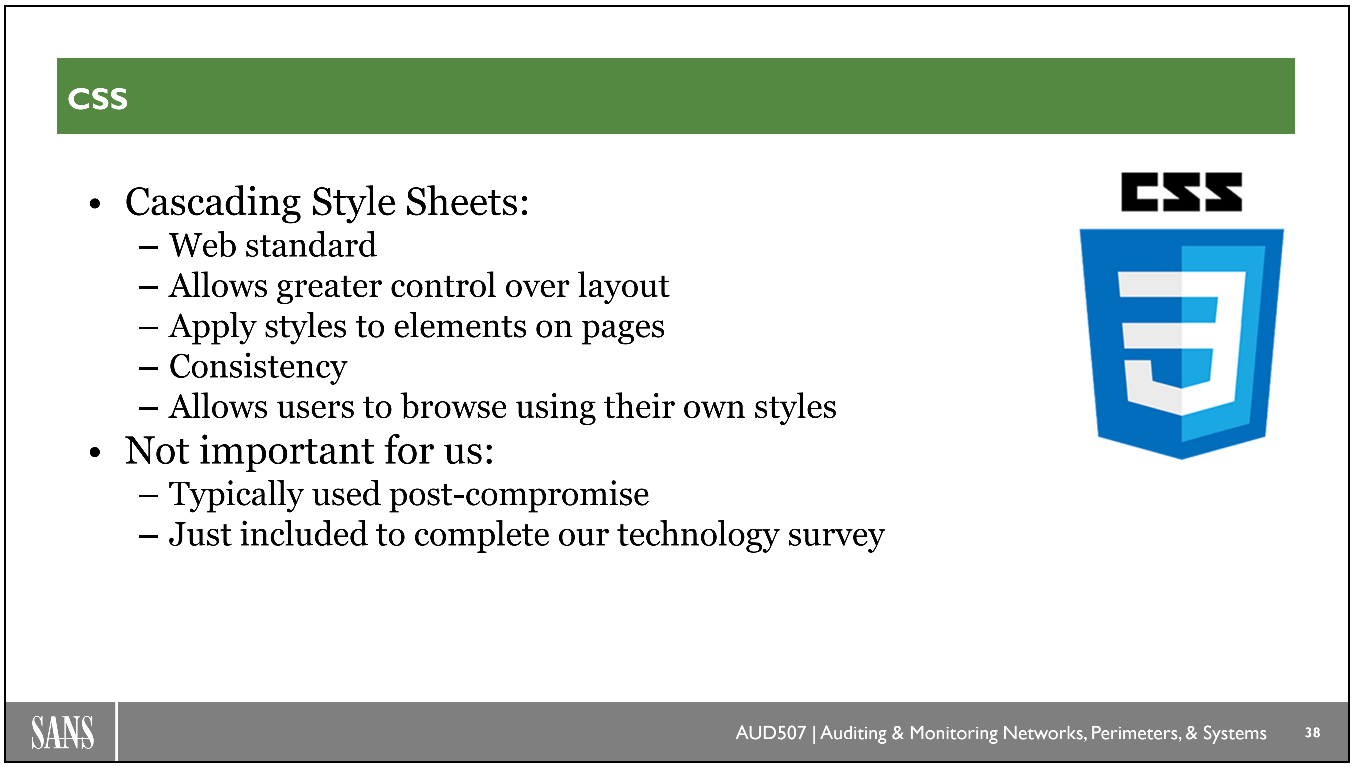
In lax mode, the browser will send the cookie for low-risk requests, but not for resource types commonly used in CSRF attacks, like images and iframes.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



CSS

- Cascading Style Sheets:
  - Web standard
  - Allows greater control over layout
  - Apply styles to elements on pages
  - Consistency
  - Allows users to browse using their own styles
- Not important for us:
  - Typically used post-compromise
  - Just included to complete our technology survey

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 38

The last of our important web-related technologies is Cascading Style Sheets (CSS). CSS has become the preferred or at least the recommended way to create portable webpages, largely supplanting the use of tables and frames.

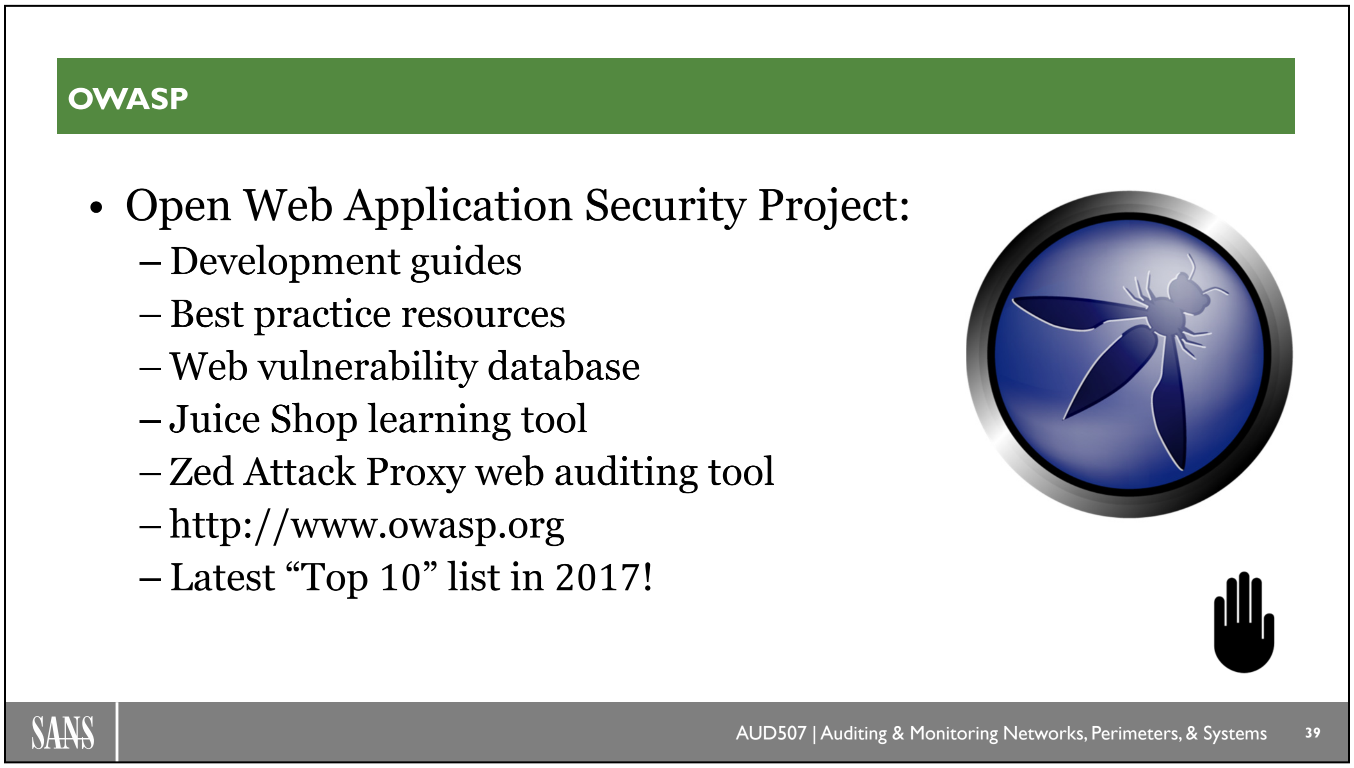
CSS is billed as a sort of cooperative language that is used with HTML. The purpose of CSS was to decouple the layout from HTML so that HTML is just used to mark up the text. Now CSS is used to control the layout of that text. All in all, this is a great idea, but there are large variations in how various browsers implement CSS. Some of them, like Internet Explorer, are known to be particularly bad when it comes to rendering CSS, requiring programmers to create CSS hacks to make their pages render properly in broken CSS environments.

Another cool feature of CSS that is not used by the majority of people is the ability to define a private style sheet on the client side and force the browser to use this style sheet for all pages that are loaded. This is primarily geared toward accessibility for persons with visual disabilities.

August 10, 2021

August 10, 2021

© SANS Institute 2021



OWASP

- Open Web Application Security Project:
  - Development guides
  - Best practice resources
  - Web vulnerability database
  - Juice Shop learning tool
  - Zed Attack Proxy web auditing tool
  - http://www.owasp.org
  - Latest "Top 10" list in 2017!

We're just about ready to give you an introduction to the primary tool that we will use to test web applications. Before we do, we want to introduce you to the Open Web Application Security Project (OWASP). OWASP is a group of security and web application professionals who banded together some years ago to help remediate some of the myriad of problems in the web application development field.

Among other things, OWASP makes available free development guides, best practice recommendations, current information on common web application vulnerabilities, and a handy web application security learning tool in Juice Shop.

While the OWASP Top 10 list is viewed as important in the industry, we are not going to make an effort in this class to align our material strongly with that list. You can be sure that you will understand what every one of those top ten issues are, but in our view many of the issues that OWASP lists as separate issues all share the same root cause. In our class, we are more focused on identifying and fixing the root causes and, as a result, treating the top ten at the same time.

August 10, 2021

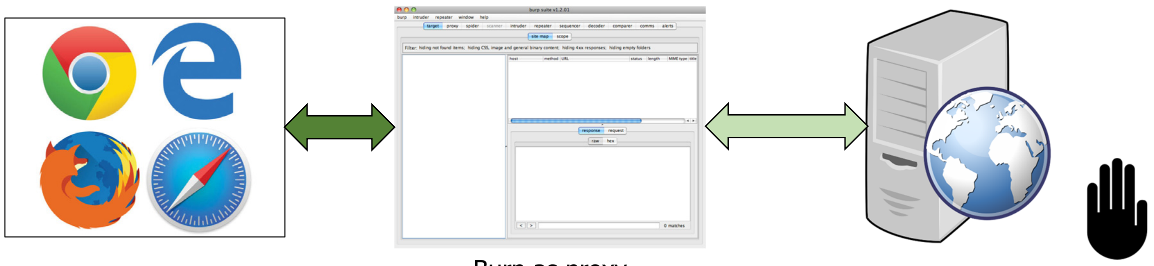
August 10, 2021

Technet24

© SANS Institute 2021

BURP Suite

- http://www.portswigger.net
  - $399 / seat / year
  - Similar to free tool: ZAP (Zed Attack Proxy)
  - Some features better, some non-existent
    - Vulnerability scanner is amazing



Burp as proxy

The primary web application testing tool that we would recommend to you is Burp Suite. Burp is a commercial tool available from PortSwigger.net for $399 per year per seat. Burp is a man-in-the-middle proxy tool. While there are free alternatives (like ZAP), Burp does some things better. One of the most outstanding features of Burp is its vulnerability scanner.

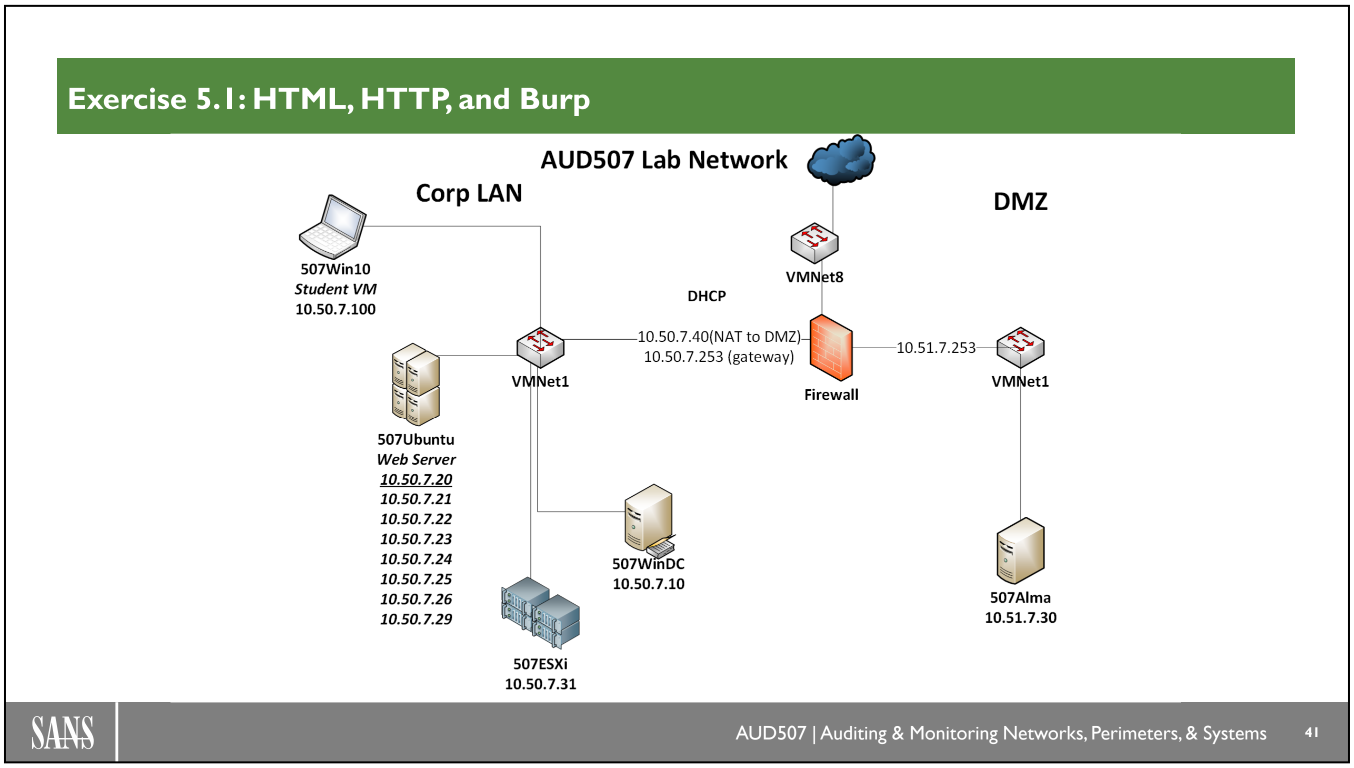
Burp includes one of the very best web application vulnerability scanners available in the industry. Not only can it passively assess pages as you browse through and interact with the web application, but it can also be configured to actively test every page and input on a target site. This is very useful for speeding up the work of the tester. Generally, if Burp tells you that it is certain that you have an issue, you can be sure that you really do have an issue.

In addition to reporting the issues that it believes that it has found, it will show you what it sent to perform its test and how the server responded, highlighting what it believes indicates there is a problem. This allows the tester to quickly validate the results.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Exercise 5.1: HTML, HTTP, and Burp

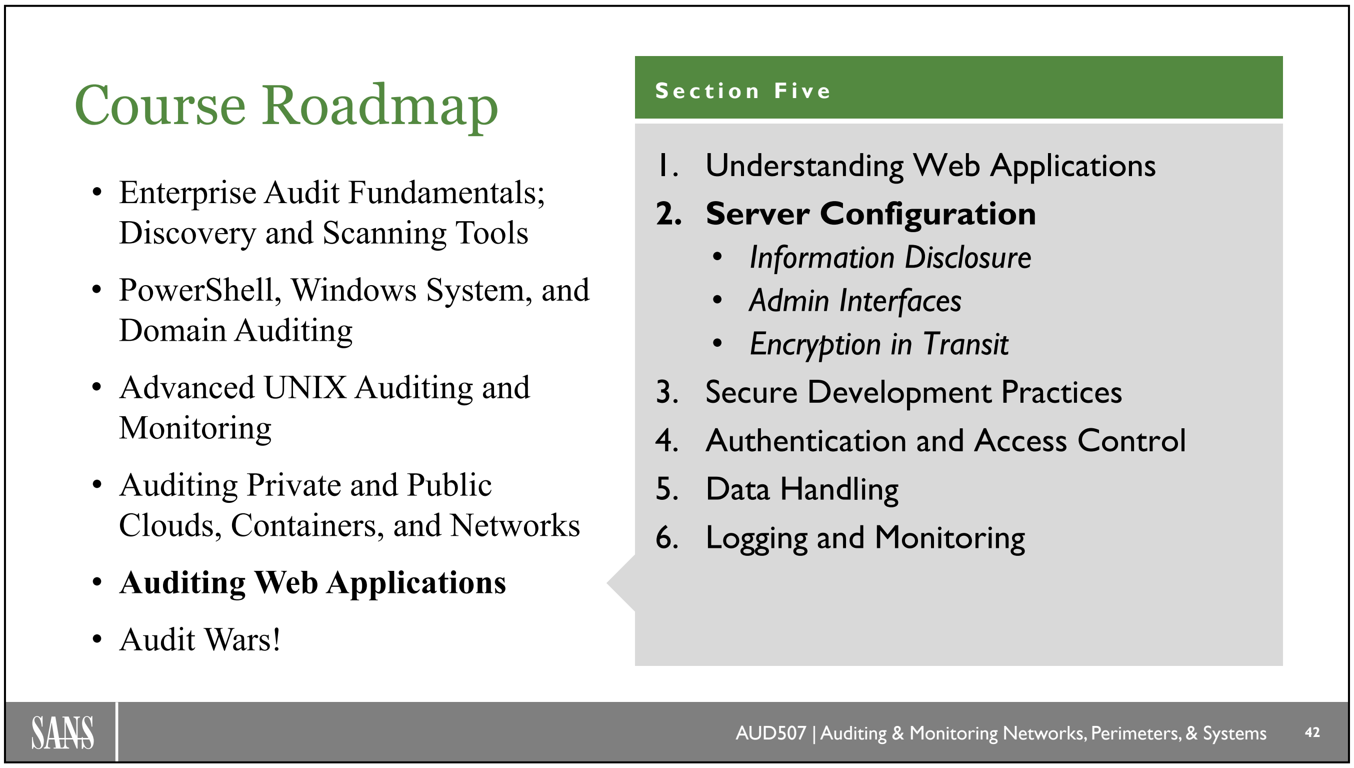
Now that we've covered the basics of HTML and HTTP, let's get started working on our exercises. Please turn to Exercise 5.1: HTML, HTTP, and Burp in your workbook!

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Course Roadmap

- Enterprise Audit Fundamentals; Discovery and Scanning Tools
- PowerShell, Windows System, and Domain Auditing
- Advanced UNIX Auditing and Monitoring
- Auditing Private and Public Clouds, Containers, and Networks
- **Auditing Web Applications**
- Audit Wars!

Section Five

1. Understanding Web Applications
2. **Server Configuration**
   - *Information Disclosure*
   - *Admin Interfaces*
   - *Encryption in Transit*
3. Secure Development Practices
4. Authentication and Access Control
5. Data Handling
6. Logging and Monitoring

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 42

This page intentionally left blank.

August 10, 2021

August 10, 2021

© SANS Institute 2021

OWASP Top Ten – 2017

- A1:2017-Injection
- A2:2017-Broken Authentication
- A3:2017-Sensitive Data Exposure
- A4:2017-XML External Entities (XXE)
- A5:2017-Broken Access Control
- A6:2017-Security Misconfiguration
- A7:2017-Cross-Site Scripting (XSS)
- A8:2017-Insecure Deserialization
- A9:2017-Using Components with Known Vulnerabilities
- A10:2017-Insufficient Logging and Monitoring

OWASP
OWASP Top 10 - 2017
The Ten Most Critical Web Application Security Risks
https://owasp.org
This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License
creative commons

The OWASP Top Ten list, which is updated every few years, is a consensus document which presents a list of the most critical vulnerabilities found in web applications. We will use the Top Ten in our audit process as a frame of reference for reporting vulnerabilities found in web applications. We don't focus our efforts on locating these vulnerabilities so much as we focus on verifying that the organization has in place the controls which commonly prevent these and other critical flaws.

August 10, 2021

August 10, 2021
Technet24

© SANS Institute 2021

OWASP Top Ten Proactive Controls

C1. Define Security Requirements
C2. Leverage Security Frameworks and Libraries
C3. Secure Database Access
C4. Encode and Escape Data
C5. Validate All Inputs
C6. Implement Digital Identity
C7. Enforce Access Controls
C8. Protect Data Everywhere
C9. Implement Security Logging and Monitoring
C10. Handle All Errors and Exceptions

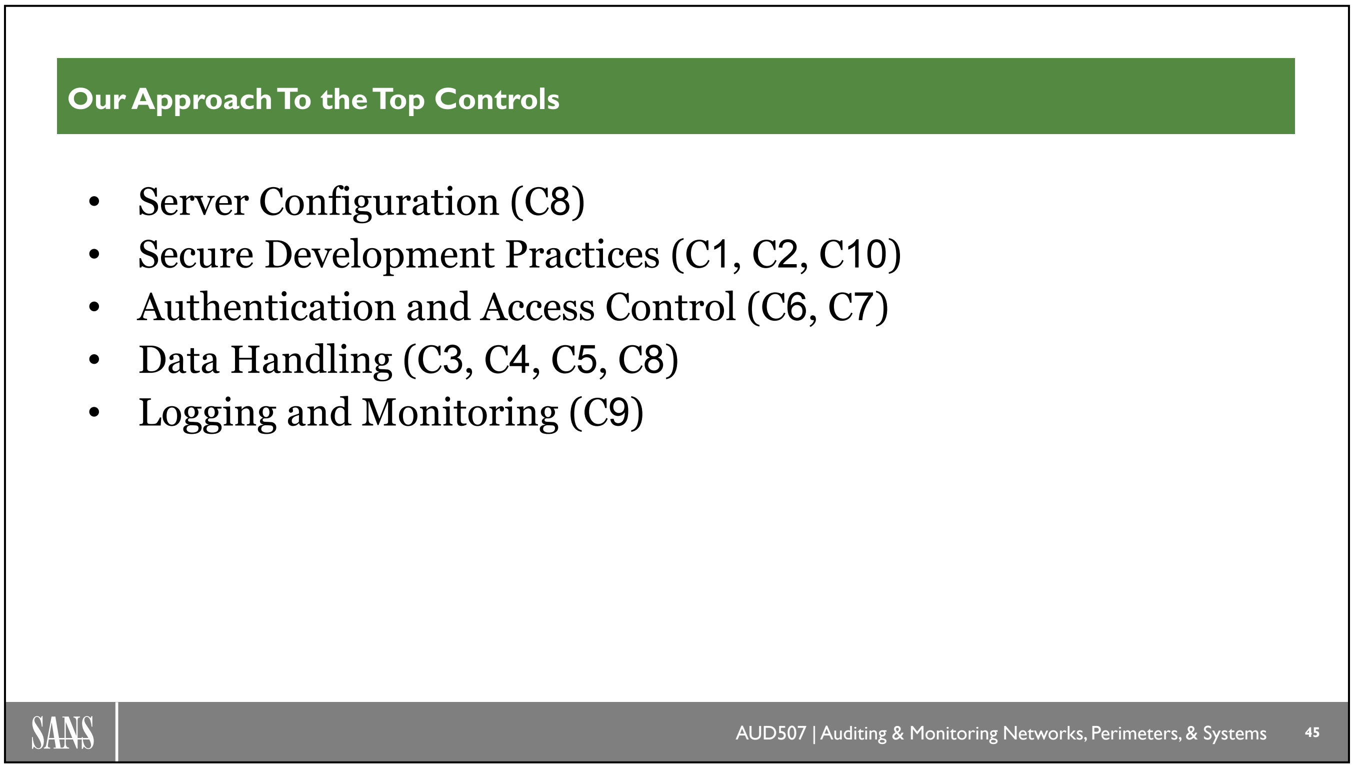
OWASP has another list of ten items which we find to be more useful to auditors – the Top Ten Proactive Controls. Version 3.0 of this list, released in 2018, defines the controls you see on this slide. We will also use this list for reference during audits, but we've distilled this list of ten controls into five control groups or security practices, which work well to cover both OWASP Top Ten lists.

The list of Proactive Controls is online at:
https://u.aud507.com/3-10

August 10, 2021

August 10, 2021

© SANS Institute 2021



Our Approach To the Top Controls

- Server Configuration (C8)
- Secure Development Practices (C1, C2, C10)
- Authentication and Access Control (C6, C7)
- Data Handling (C3, C4, C5, C8)
- Logging and Monitoring (C9)

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 45

Our list of five control groups gives us the framework we need to design and perform web application audits. These secure development practices, when implemented properly by the organization, will serve to protect against the vulnerabilities on the OWASP Top Ten, and to satisfy the Proactive Controls.

Here's a simple example:

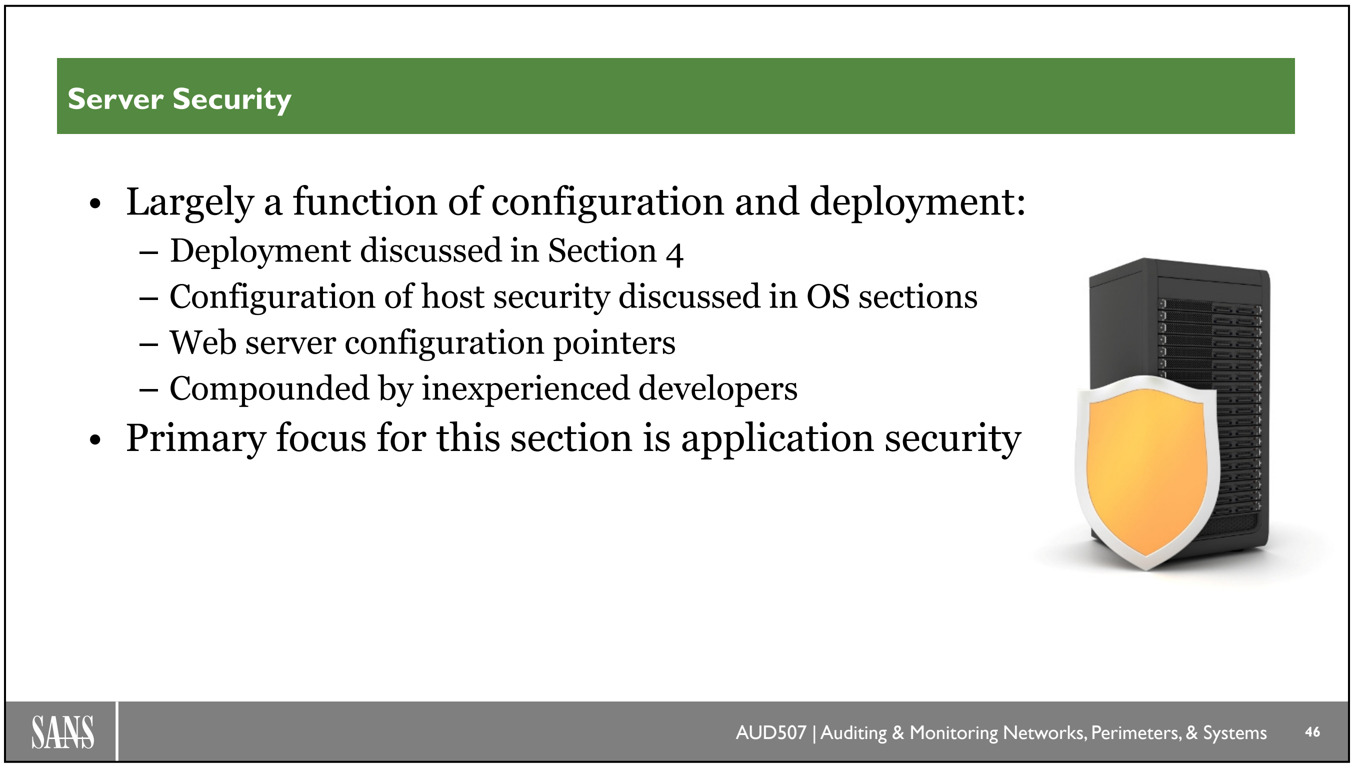
A4:2017-XML External Entities (XXE) from the Top Ten describes ways an attacker can inject content into transactions being performed by web services which exchange data using XML. The Proactive Controls C3, C4, C5, and C8 are all controls over data handling which can be used together to defend against XXE injection. We treat those controls as a single development practice of handling data properly.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Server Security

- Largely a function of configuration and deployment:
  - Deployment discussed in Section 4
  - Configuration of host security discussed in OS sections
  - Web server configuration pointers
  - Compounded by inexperienced developers
- Primary focus for this section is application security

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 46

As we begin this section, we want to be clear that we have no intention of talking about all the myriad of possible server security options for a web server. We also do not plan to talk about how to deploy a web server securely in a network, nor will we talk about securing the underlying operating system of the host.

Ample security configuration guides for the various web servers are on the market today (and we'll give you pointers to a few good ones), so it is not a wise use of our time to focus on these. In the second and third sections of this course we covered secure network deployment and firewall placement, so we will not rehash those topics. Finally, we talk in great detail about host-level security during the Windows and Unix sections, so we will, again, not duplicate material. We'll also see examples that illustrate how simple misconfiguration issues can be compounded by silly developer tricks, creating enormous vulnerability for the organization.

The primary focus of this entire section is on the web applications. This is where we find that organizations have the biggest weaknesses, especially because these applications are often deployed to the public. In this section, we look primarily at server configuration issues, but we also pull in some application development issues and try to illustrate how simple misconfigurations coupled with poor application practices combine to create big problems.

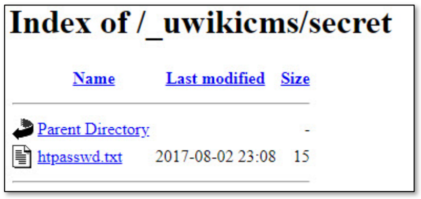
August 10, 2021

August 10, 2021

© SANS Institute 2021

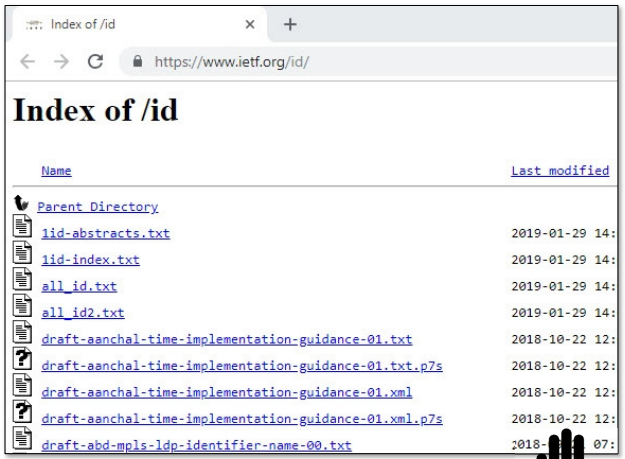
Directory Indexing

- More of a disclosure than a vulnerability:
  - Note the URL
  - Access a directory rather than a webpage
  - Might be intentional
- Configurable on all major web servers:
  - Generally, default is to load index.htm or index.html
  - Check if setting matches security stance and purpose



Index of /_uwikicms/secret

```
Name            Last modified     Size
Parent Directory                    -
htpasswd.txt    2017-08-02 23:08   15
```



https://www.ietf.org/id/

Index of /id

```
Name                                                    Last modified
Parent Directory
1id-abstracts.txt                                       2019-01-29 14:
1id-index.txt                                           2019-01-29 14:
all_id.txt                                              2019-01-29 14:
all_id2.txt                                             2019-01-29 14:
draft-aanchal-time-implementation-guidance-01.txt       2018-10-22 12:
draft-aanchal-time-implementation-guidance-01.txt.p7s   2018-10-22 12:
draft-aanchal-time-implementation-guidance-01.xml       2018-10-22 12:
draft-aanchal-time-implementation-guidance-01.xml.p7s   2018-10-22 12:
draft-abd-mpls-ldp-identifier-name-00.txt               2018-     07:
```

The first on our list is directory indexing, which is an example of your web server just trying to be helpful. All web servers enable you to configure a default page to be loaded when someone sends a request to the site but does not request a specific page. How could this be? Well, consider what you type to go to the Internet Engineering Task Force (IETF) website: https://www.ietf.org. Although you have clearly requested a webpage, what is the name of that page? Because you didn't specify a name, the web server automatically looks for whatever it has been configured to consider the default page.

What would happen if there were no default page to load? This is where directory indexing comes in. On the Internet Drafts portion of the IETF site, directory indexing is intentionally turned on, and you can see the effect of it when you go to the www.ietf.org/id/ directory. Because there is no default page, the web server displays the contents of the directory instead.

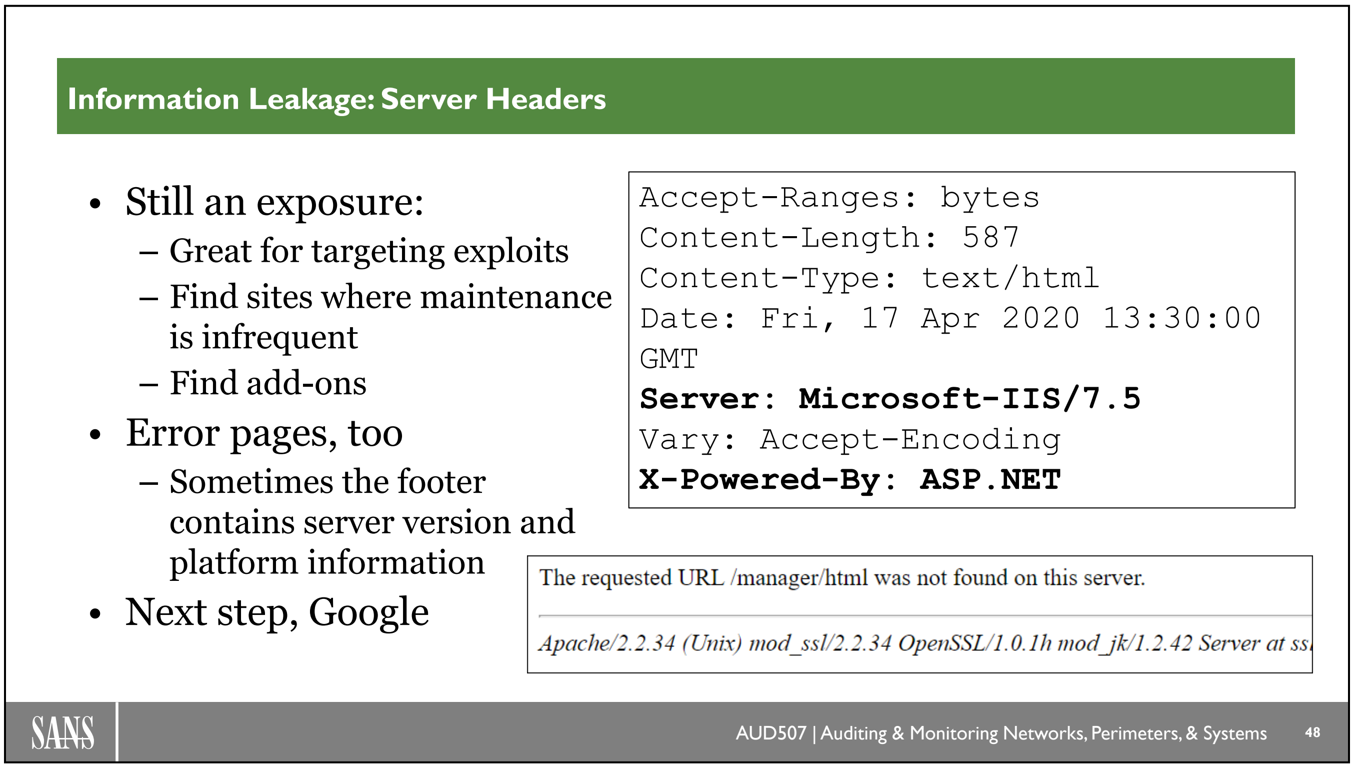
Although this is handy in the IETF case, it can have some unintended consequences for security. Note the contents of the other screenshot on this slide. Do you think the administrator intended for this directory's contents to be made public?

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Information Leakage: Server Headers

- Still an exposure:
  - Great for targeting exploits
  - Find sites where maintenance is infrequent
  - Find add-ons
- Error pages, too
  - Sometimes the footer contains server version and platform information
- Next step, Google

```
Accept-Ranges: bytes
Content-Length: 587
Content-Type: text/html
Date: Fri, 17 Apr 2020 13:30:00
GMT
Server: Microsoft-IIS/7.5
Vary: Accept-Encoding
X-Powered-By: ASP.NET
```

```
The requested URL /manager/html was not found on this server.

Apache/2.2.34 (Unix) mod_ssl/2.2.34 OpenSSL/1.0.1h mod_jk/1.2.42 Server at ss
```

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 48

While we're on the topic of exposures, let's talk about headers for a few moments. Earlier we spoke about HTTP headers briefly. We focused only on the Cookie header. Notice in the slide though that there is a lot more information in the headers than just the cookies. In this case, we are looking at three things.

In the bottom picture, we can see the footer on a webpage that was loaded on the internet. Notice that the footer tells you precisely the version of the web server that is running. Even without this footer, notice the other screenshot. Here we can see an IIS server. Further, we can identify add-on features like mod_ssl for the Apache server.

Again, we are looking at an exposure here. In addition to giving us wonderful targeting information, allowing us to simply search for exploits that work on the version of the server running, it also gives us a good feel for how well maintained the site is. How so? Well, if the Apache version is three years old, do you think that it's likely that anyone is watching in terms of security and monitoring?

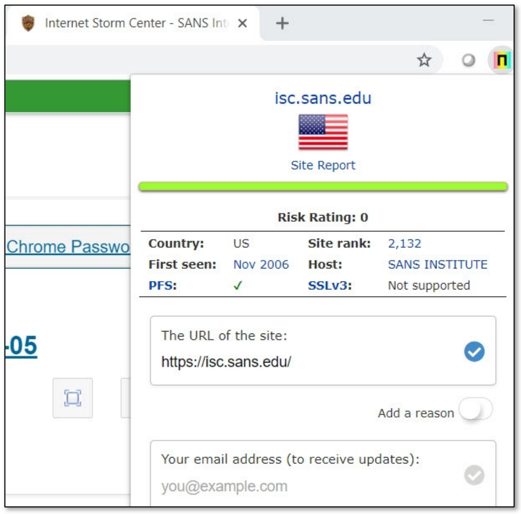
August 10, 2021

August 10, 2021

© SANS Institute 2021

Netcraft Toolbar

- Available as browser plugin, or at toolbar.netcraft.com
- Tracks server type, TLS information, technologies in use and more for websites
- Use the "site report" function to see the site's history



Netcraft is an internet security vendor who maintains information about the technologies used on web servers. If you've ever seen statistics describing what percentage of the servers on the web are running Apache or IIS, they may well have been developed by Netcraft.

The Netcraft toolbar and its associated website (https://u.aud507.com/3-11) allow the auditor to view current and historical information about the technologies in use on a site, including the web server version, content management systems, TLS versions, and certificate information. This data can be accessed either through the browser plugin or directly from the Netcraft website.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Robots.txt

- Intended to control indexing:
  - Configure restrictions based on web crawlers
  - Common to give away extra information

```
User-agent: *
Disallow: /annual-report.html
Disallow: /financials.html
Disallow: /donate.html
Disallow: /proposalform.html
```

Another configuration issue revolves around the robots file. This file has been around for quite some time and is intended to enable you to control what parts of your site a well-behaved web crawler (such as Google or Yahoo) indexes. More specifically, it enables you to specify sections of your site that should *not* be indexed by search engines.

There is a side effect of using this file, however. Some organizations list sensitive portions of their site in the robots file, essentially telling an attacker where the most important parts of your site are. Now, we're not saying that you shouldn't use a robots file. Certainly, it has a useful function. What we are saying is that you should look carefully at what you are listing in your robots file. It is not uncommon to find pointers to nonpublic pieces of the site such as remote webmail, remote desktop connections, and more.

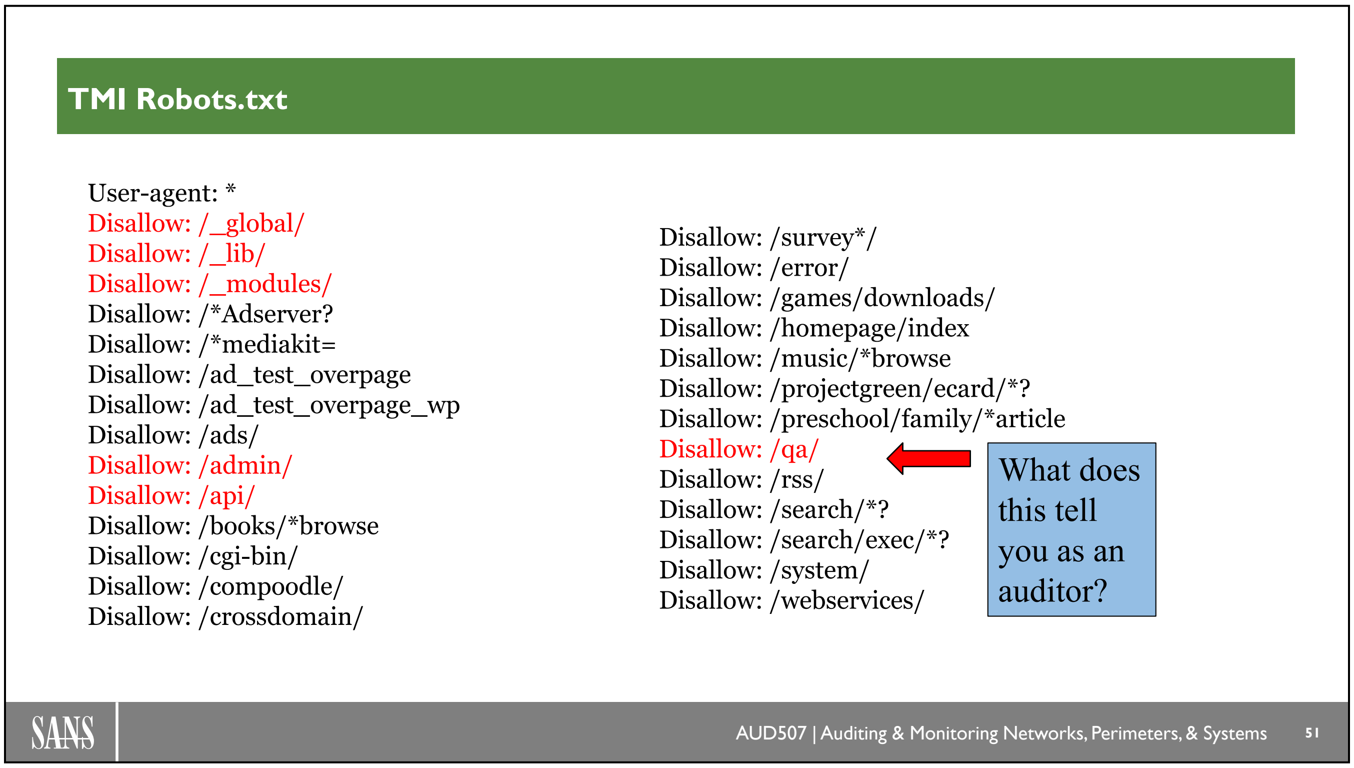
If you want to restrict which pages are indexed and you want to avoid putting something in the robots file, this can be accomplished more cleanly using META tags marking the content as "private." An even better solution is to never put sensitive information into the public areas of your website!

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



TMI Robots.txt

```
User-agent: *
Disallow: /_global/
Disallow: /_lib/
Disallow: /_modules/
Disallow: /*Adserver?
Disallow: /*mediakit=
Disallow: /ad_test_overpage
Disallow: /ad_test_overpage_wp
Disallow: /ads/
Disallow: /admin/
Disallow: /api/
Disallow: /books/*browse
Disallow: /cgi-bin/
Disallow: /compoodle/
Disallow: /crossdomain/
Disallow: /survey*/
Disallow: /error/
Disallow: /games/downloads/
Disallow: /homepage/index
Disallow: /music/*browse
Disallow: /projectgreen/ecard/*?
Disallow: /preschool/family/*article
Disallow: /qa/
Disallow: /rss/
Disallow: /search/*?
Disallow: /search/exec/*?
Disallow: /system/
Disallow: /webservices/
```

What does this tell you as an auditor?

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 51

Here's an example from a real corporation. There are still errors in the way that this site is built from a security perspective.

Looking at the directories that have been "excluded" from indexing, what sorts of things do you think you might find in these directories? Of all the things that are in the slide, the qa directory is by far the most interesting. What does that directory seem to indicate about the environment where development occurs? If I were to find this on a production site, it would be a strong indication to me that the production site is also used for Quality Assurance. This is certainly a bad idea!

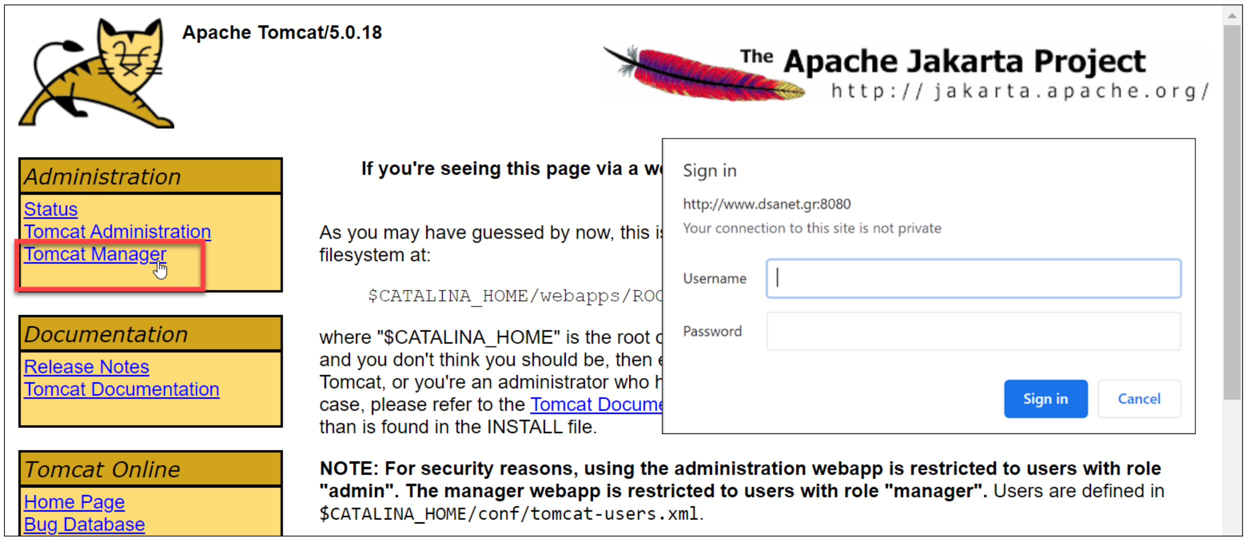
August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Publicly Exposed Admin Interface



Apache Tomcat/5.0.18

If you're seeing this page via a w

As you may have guessed by now, this i
filesystem at:

```
$CATALINA_HOME/webapps/ROO
```

where "$CATALINA_HOME" is the root o
and you don't think you should be, then e
Tomcat, or you're an administrator who h
case, please refer to the Tomcat Docume
than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Sign in
http://www.dsanet.gr:8080
Your connection to this site is not private

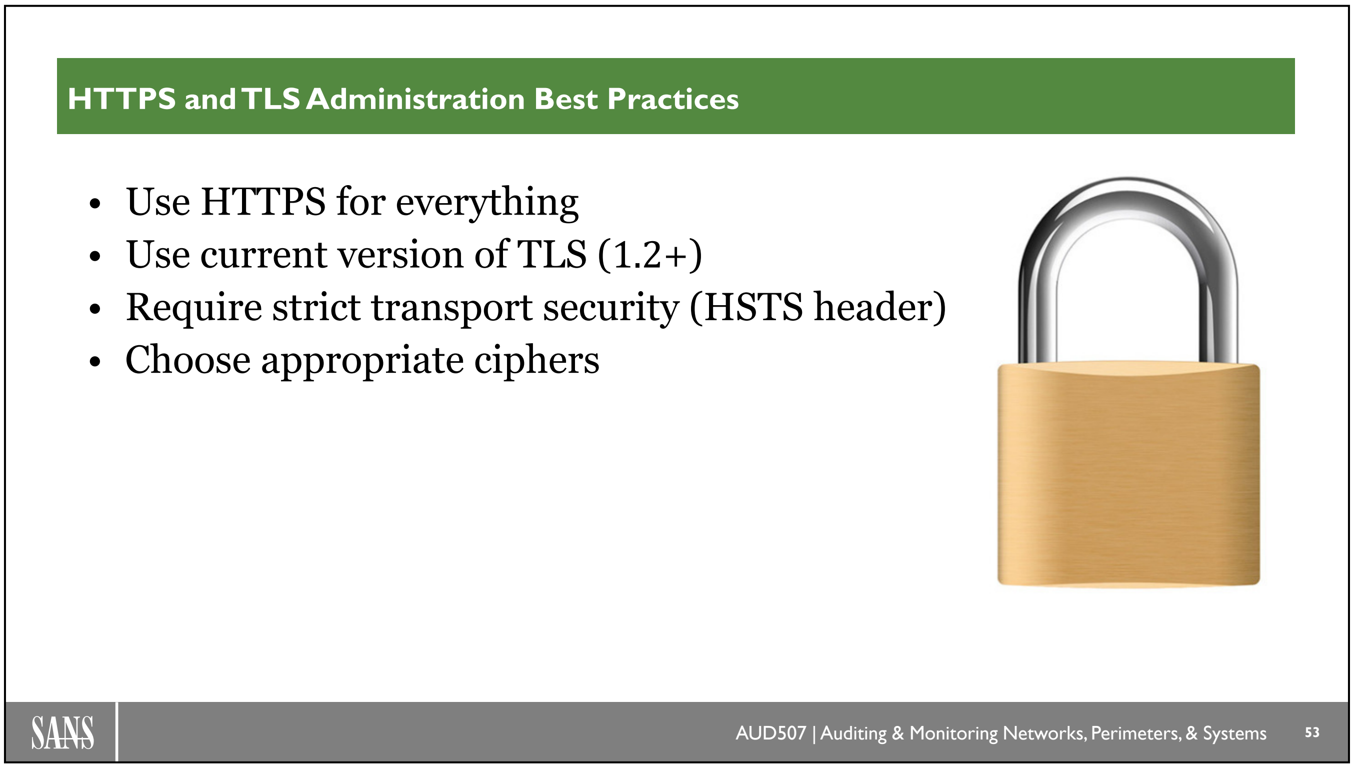
A problem we commonly see is web-based administrative interfaces installed and configured to be available to the general public. In this screenshot, the web administrators have installed the Apache Tomcat application server and left it configured to host its default page to the world. We can also see that the Tomcat Manager application is installed. Clicking on the link causes the browser to produce a pop-up dialog box for HTTP basic authentication. Basic authentication has no lockout mechanism by default, meaning that an attacker could potentially perform a brute-force attack to gain control of this application server. From there, she could change the applications hosted on the server, pivot to steal customer data, or simply delete the application and settings to render the server unavailable.

It is critical that, whenever possible, administrative interfaces are protected against public access.

August 10, 2021

August 10, 2021

© SANS Institute 2021



HTTPS and TLS Administration Best Practices

- Use HTTPS for everything
- Use current version of TLS (1.2+)
- Require strict transport security (HSTS header)
- Choose appropriate ciphers

Many websites today are using HTTPS for all web traffic. This approach has advantages. Some attacks against users simply will not work with HTTPS (credential and cookie sniffing, for example). Users can see the oh-so-important padlock in the address bar, which lets them know that your site is trustworthy. With the advent of Certificate Authorities like Let's Encrypt, valid certificates can now be obtained for free.

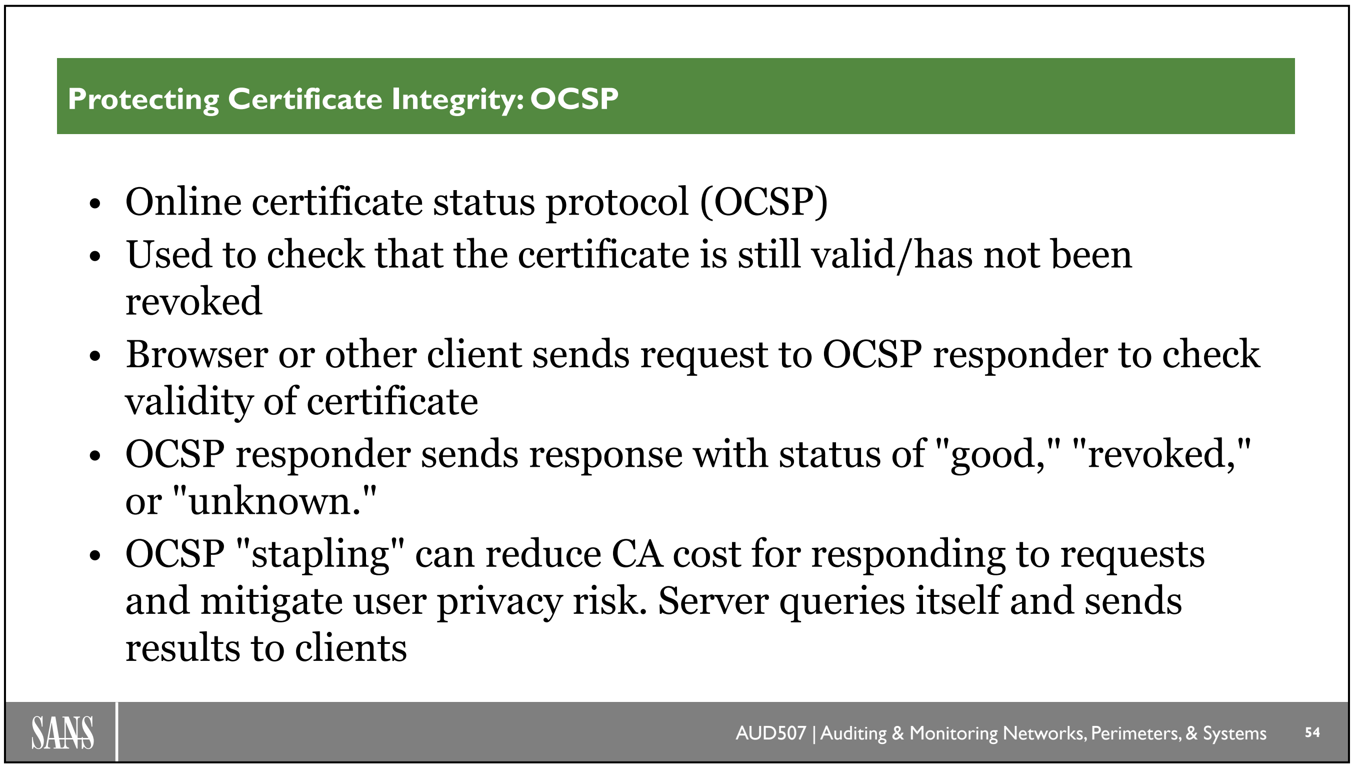
When configuring a server for HTTPS, the server should use the most recent versions of TLS. As of this writing, version 1.2 is commonly deployed, but version 1.3 is available. Sites should move to use the most modern version which can be supported by their clients.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Protecting Certificate Integrity: OCSP

- Online certificate status protocol (OCSP)
- Used to check that the certificate is still valid/has not been revoked
- Browser or other client sends request to OCSP responder to check validity of certificate
- OCSP responder sends response with status of "good," "revoked," or "unknown."
- OCSP "stapling" can reduce CA cost for responding to requests and mitigate user privacy risk. Server queries itself and sends results to clients

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 54

Protocols are now available to prove that the server's security certificate is valid. The online certificate status protocol (OCSP) allows a browser to check to see that the server certificate has not been revoked for any reason. Under OCSP, the client (usually a web browser) makes a request to an OCSP responder (usually run by the certificate authority) to validate the status of a certificate presented by the webserver. Response codes will indicate whether the certificate' status is good, revoked or unknown.

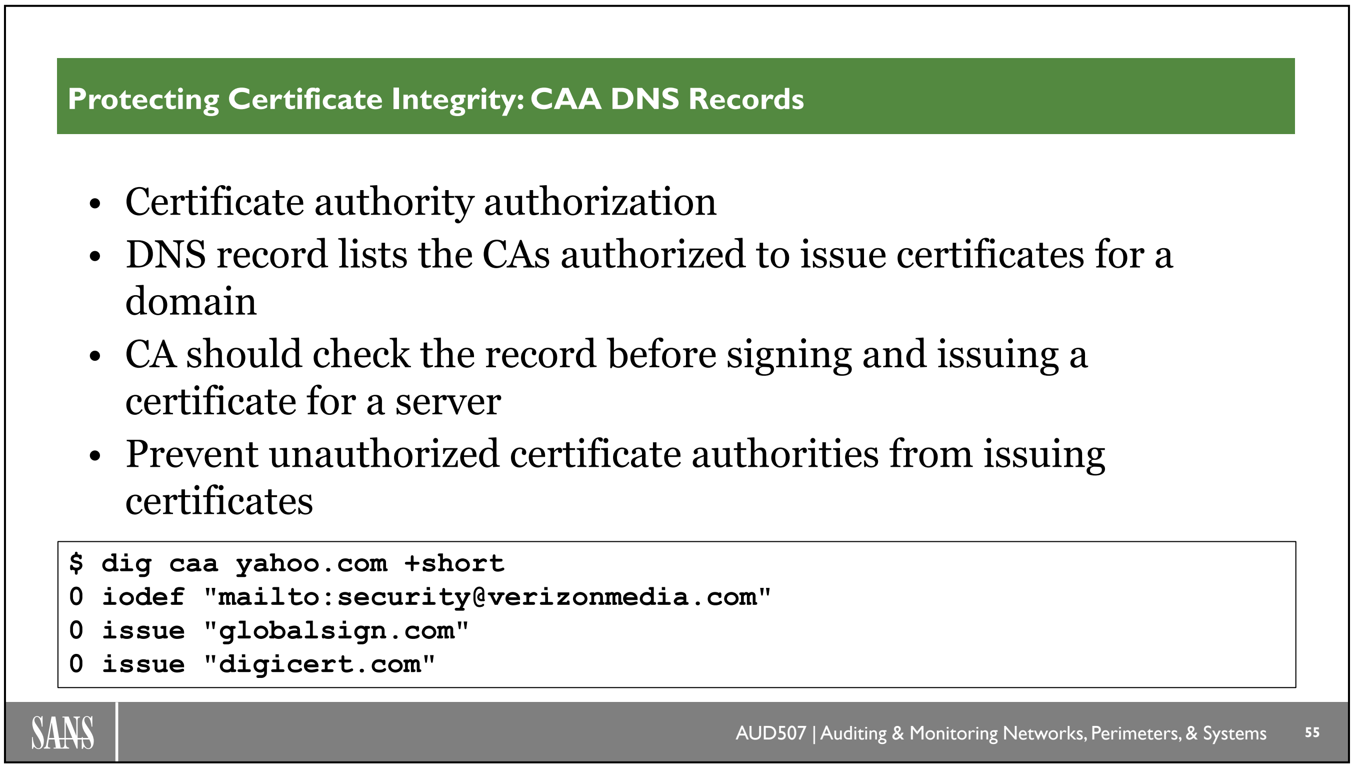
As you can imagine, this protocol can put quite a load on the responder, if every client is querying for the certificate status for every server using one of the CA's certificates. Additionally, there is a privacy concern that the CAs could know which websites a user visits by tracking the requests received. To overcome these limitations, OCSP "stapling" involves the server querying for its own certificate status and then sending the digitally signed results to the client during TLS negotiation. This must be explicitly requested by the client, but modern browsers all support this.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Protecting Certificate Integrity: CAA DNS Records

- Certificate authority authorization
- DNS record lists the CAs authorized to issue certificates for a domain
- CA should check the record before signing and issuing a certificate for a server
- Prevent unauthorized certificate authorities from issuing certificates

```
$ dig caa yahoo.com +short
0 iodef "mailto:security@verizonmedia.com"
0 issue "globalsign.com"
0 issue "digicert.com"
```

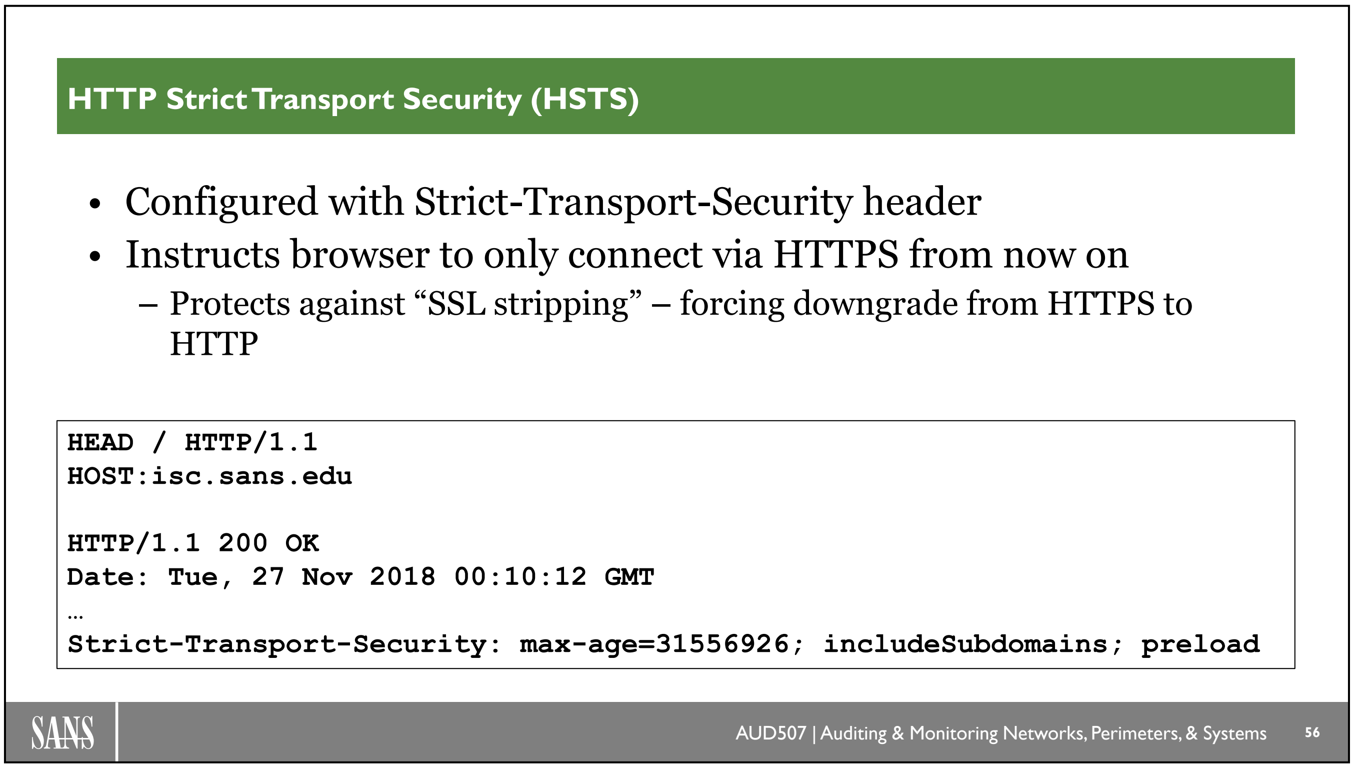
Certificate Authority Authorization (CAA) records in DNS specify exactly which internet Certificate Authorities (CAs) are authorized to issue certificates for a domain. A certificate signing request to any certificate authority other than those listed in the CAA record should be rejected. This helps to protect against impersonation using certificates issued by a legitimate CA which were obtained by fraud.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



HTTP Strict Transport Security (HSTS)

- Configured with Strict-Transport-Security header
- Instructs browser to only connect via HTTPS from now on
  - Protects against “SSL stripping” – forcing downgrade from HTTPS to HTTP

```
HEAD / HTTP/1.1
HOST:isc.sans.edu

HTTP/1.1 200 OK
Date: Tue, 27 Nov 2018 00:10:12 GMT
…
Strict-Transport-Security: max-age=31556926; includeSubdomains; preload
```

SANS

AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 56

The HTTP “Strict-Transport-Security” (HSTS) header tells the browser only to use HTTPS to communicate with this site from now on. This header helps to protect against “SSL stripping” attacks, in which an attacker uses an HTTP redirect to force a browser to load the HTTP version of a site, thus allowing easier sniffing of cookies and credentials.

The HSTS header can include a “Max-Age” attribute, specifying how long (in seconds) the browser should honor the HSTS settings. The recommended Max-Age is at least one year for most sites.

The “pre-load” attribute asks browser developers, like Microsoft and Google, etc., to add this site to their pre-loaded list of HTTPS-only sites which they include with their browser.

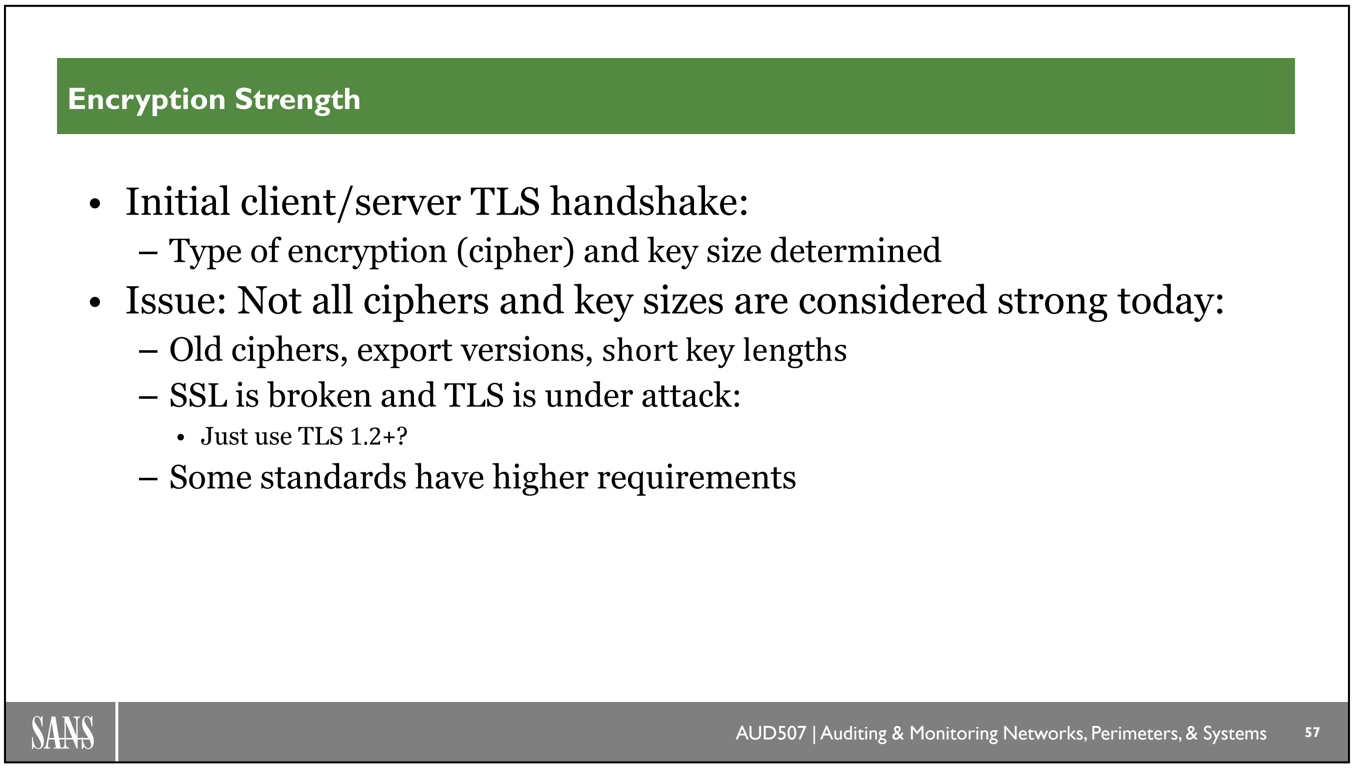
Once a browser has received the HSTS header, it will ONLY load the site with HTTPS until the Max-Age has expired.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Encryption Strength

- Initial client/server TLS handshake:
  - Type of encryption (cipher) and key size determined
- Issue: Not all ciphers and key sizes are considered strong today:
  - Old ciphers, export versions, short key lengths
  - SSL is broken and TLS is under attack:
    - Just use TLS 1.2+?
  - Some standards have higher requirements

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 57

When we say "encryption" we are actually talking about several things, including a key exchange, a cryptographic cipher, and the key size used by the cipher. All this and more are negotiated during the initial connection made between a web browser and server.

The problem is that not all ciphers are as strong as others. The web server may be offering weak ciphers to the users of a web application with highly sensitive data. Another extremely important problem is that we may not be following the "best practice" of disabling everything below TLS 1.2.

Why can't we just enforce TLS 1.2 everywhere? After all, the majority (though not all) of the TLS and SSL issues in recent memory (POODLE, BREACH, Heartbleed…) take advantage of weaknesses in older versions of these standards or use a man in the middle to force two partners to downgrade the cipher suite that will be used. Requiring TLS 1.2 and disabling all weak ciphers completely should solve the problem, right?

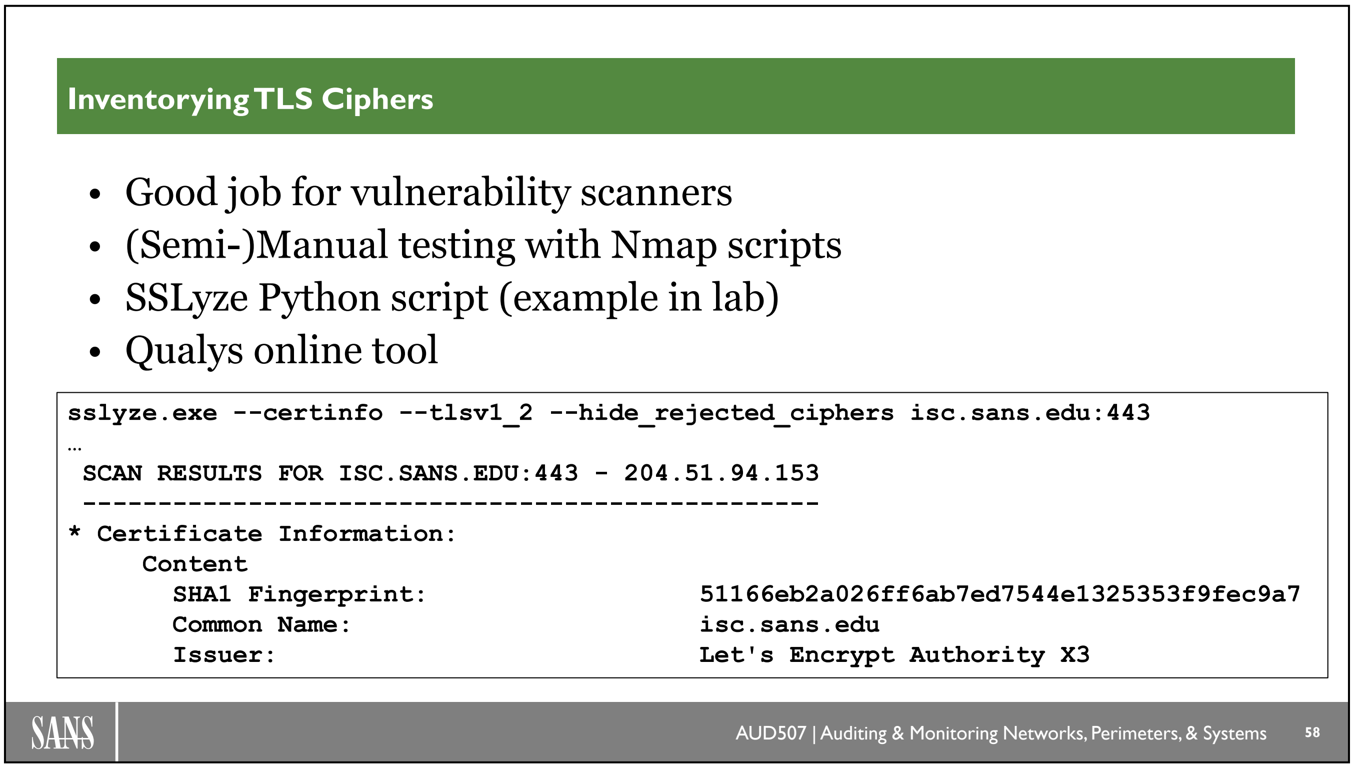
The big issue here is that we run straight into operational issues. We cannot guarantee that every customer who accesses our site will have a browser that supports TLS 1.2. Worse, we can't guarantee that all our customers can (legally) support the best encryption algorithms or key lengths! What all this means is that we must understand the operational requirements, research current threats, and provide advice to the business based on our research and the deployment that is in place.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Inventorying TLS Ciphers

- Good job for vulnerability scanners
- (Semi-)Manual testing with Nmap scripts
- SSLyze Python script (example in lab)
- Qualys online tool

```
sslyze.exe --certinfo --tlsv1_2 --hide_rejected_ciphers isc.sans.edu:443
…
 SCAN RESULTS FOR ISC.SANS.EDU:443 - 204.51.94.153
 --------------------------------------------------
* Certificate Information:
     Content
       SHA1 Fingerprint:                  51166eb2a026ff6ab7ed7544e1325353f9fec9a7
       Common Name:                       isc.sans.edu
       Issuer:                            Let's Encrypt Authority X3
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 58

A standard part of a web server configuration audit should be to inventory the TLS versions and cipher suites which are offered by the server. Most vulnerability scanners perform this task reasonably well. Nmap has Nmap scripting engine (NSE) scripts included which can gather certificate information and inventory the ciphers available. SSLyze is a Python script which can inventory ciphers and protocols, as well as testing for certain protocol vulnerabilities, like Heartbleed and Poodle attacks. Qualys has a nice online tool (shown on the next two slides) which can perform good inventories of servers which face the internet.

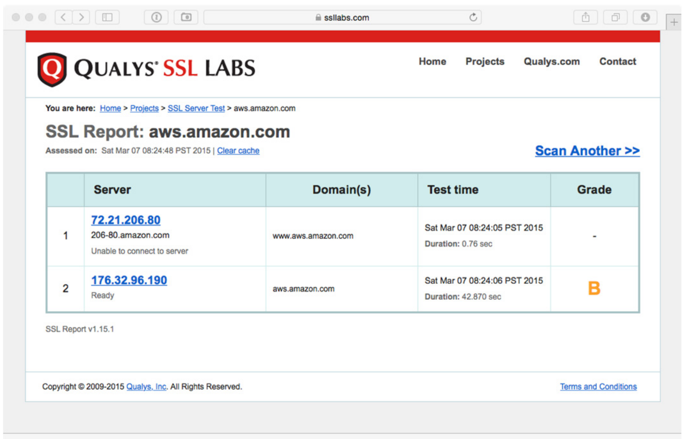
August 10, 2021

August 10, 2021

© SANS Institute 2021

Automated TLS Testing

- Qualys has a free service: https://www.ssllabs.com/ssltest/



You are here: Home > Projects > SSL Server Test > aws.amazon.com

SSL Report: aws.amazon.com

Assessed on: Sat Mar 07 08:24:48 PST 2015 | Clear cache

Scan Another >>

	Server	Domain(s)	Test time	Grade
1	72.21.206.80 206-80.amazon.com Unable to connect to server	www.aws.amazon.com	Sat Mar 07 08:24:05 PST 2015 Duration: 0.76 sec	-
2	176.32.96.190 Ready	aws.amazon.com	Sat Mar 07 08:24:06 PST 2015 Duration: 42.870 sec	B

SSL Report v1.15.1

Copyright © 2009-2015 Qualys, Inc. All Rights Reserved.

Terms and Conditions

Qualys has a free tool to help you evaluate the configuration of your TLS settings, certificates, and key lengths. Simply enter the address of the site to be tested and off it goes!

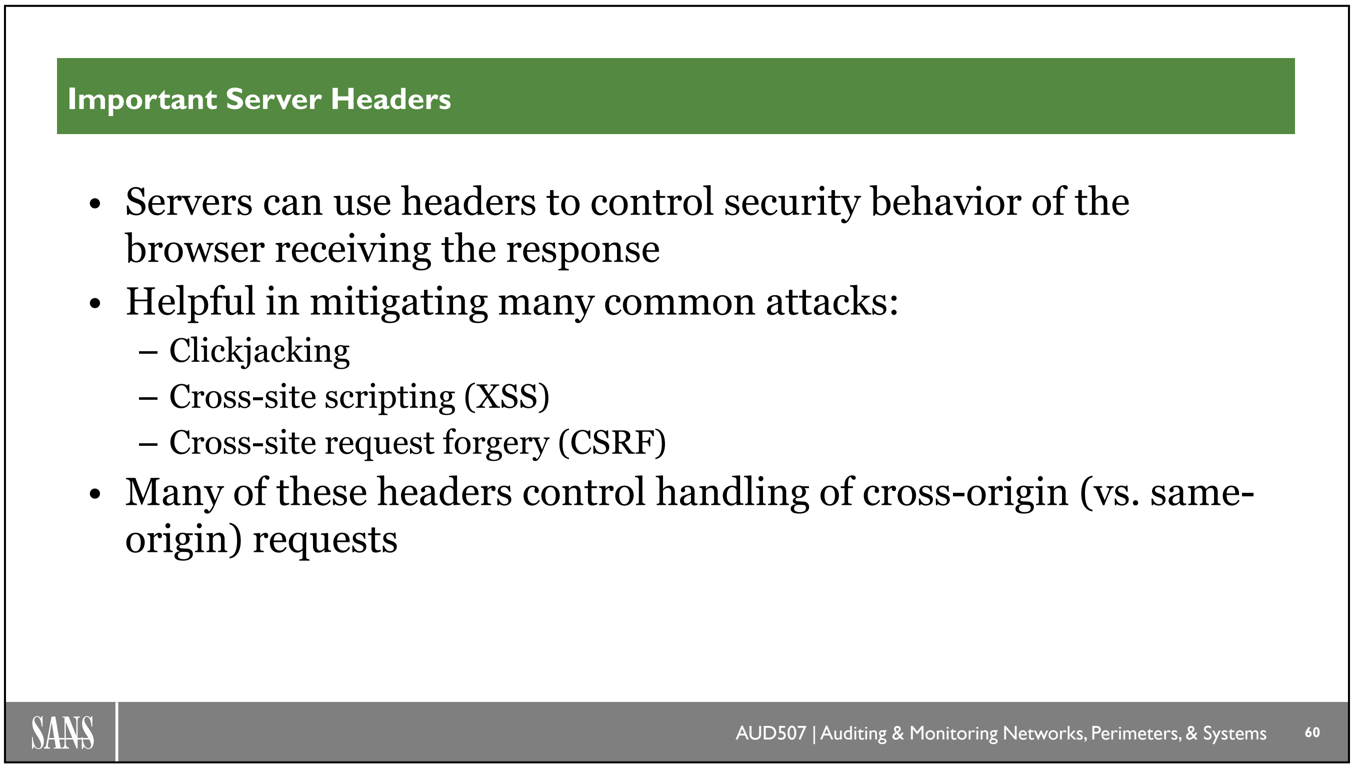
After approximately one minute, the assessment will be complete, and you will receive a letter grade for the site. What does the letter grade mean? Simply click the IP address of the host on the left side to find out.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Important Server Headers

- Servers can use headers to control security behavior of the browser receiving the response
- Helpful in mitigating many common attacks:
  - Clickjacking
  - Cross-site scripting (XSS)
  - Cross-site request forgery (CSRF)
- Many of these headers control handling of cross-origin (vs. same-origin) requests

AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 60

No discussion of server configuration would be complete without covering some of the other HTTP headers available to enhance the security of our web applications. Over the years, many headers have been created to allow the server to inform the browser as to what is or is not allowed when interacting with the website.

These headers help protect against attacks like:

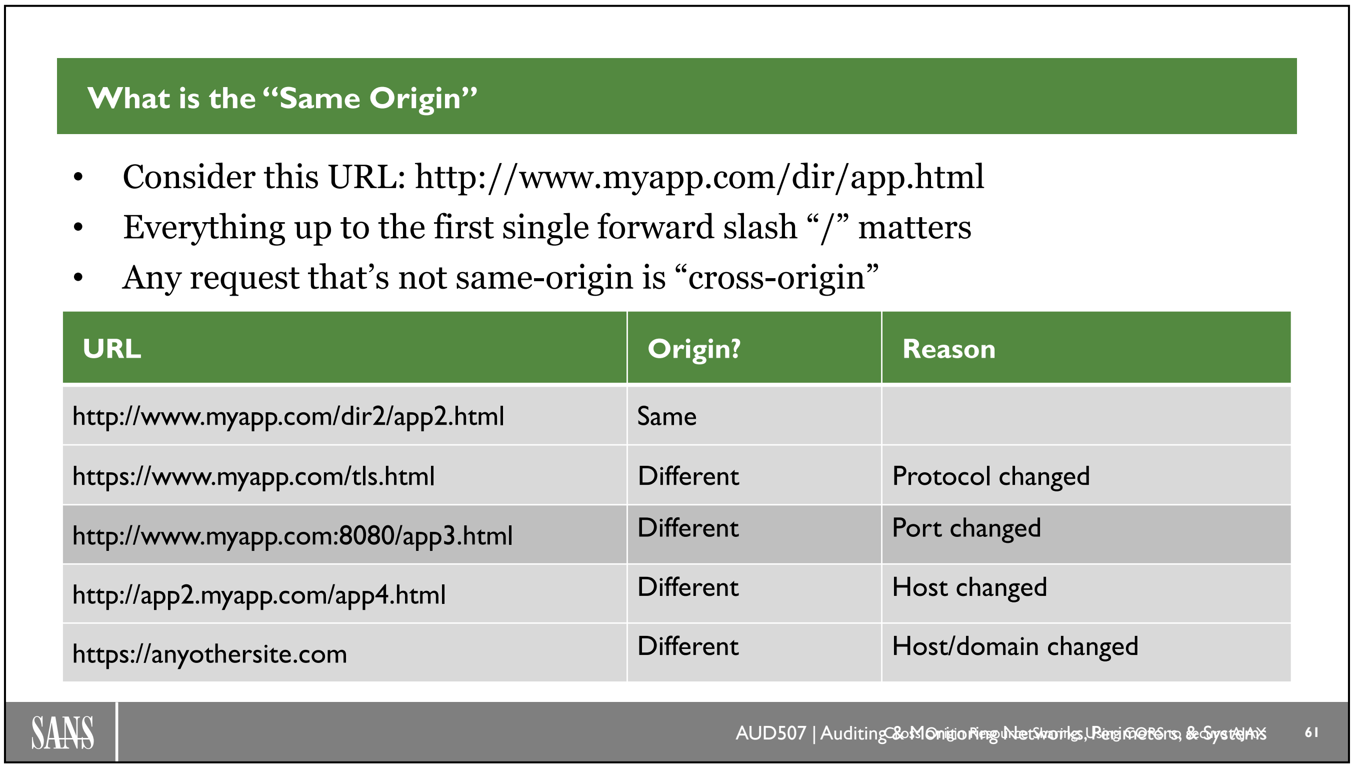
- Clickjacking: embedding part of a victim website into a transparent iframe on an evil site in order to trick the user into clicking something on the victim site
- Cross-site scripting (or script injection): using malicious or malformed input to “inject” a script into a user’s browser to be executed
- Cross-site request forgery: tricking a victim browser into making a request to a third-party site, often with the user's session credentials included

On the next several slides, we will discuss some of these headers, their intended use, and what to look for during your web application audits. First, we will discuss the meaning of a same-origin request.

August 10, 2021

August 10, 2021

© SANS Institute 2021



What is the "Same Origin"

- Consider this URL: http://www.myapp.com/dir/app.html
- Everything up to the first single forward slash "/" matters
- Any request that's not same-origin is "cross-origin"

URL	Origin?	Reason
http://www.myapp.com/dir2/app2.html	Same	
https://www.myapp.com/tls.html	Different	Protocol changed
http://www.myapp.com:8080/app3.html	Different	Port changed
http://app2.myapp.com/app4.html	Different	Host changed
https://anyothersite.com	Different	Host/domain changed

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 61

Browsers see cross-origin requests as riskier than same-origin requests. After all, most developers will not intentionally attack their own site, but they might attack someone else's. The rules for determining whether a request is cross-origin are pretty simple. Basically, if everything before the first forward-slash of the URL matches between the page I'm on and the page I'm about to request, then the request is same-origin and will be subjected to much less scrutiny.

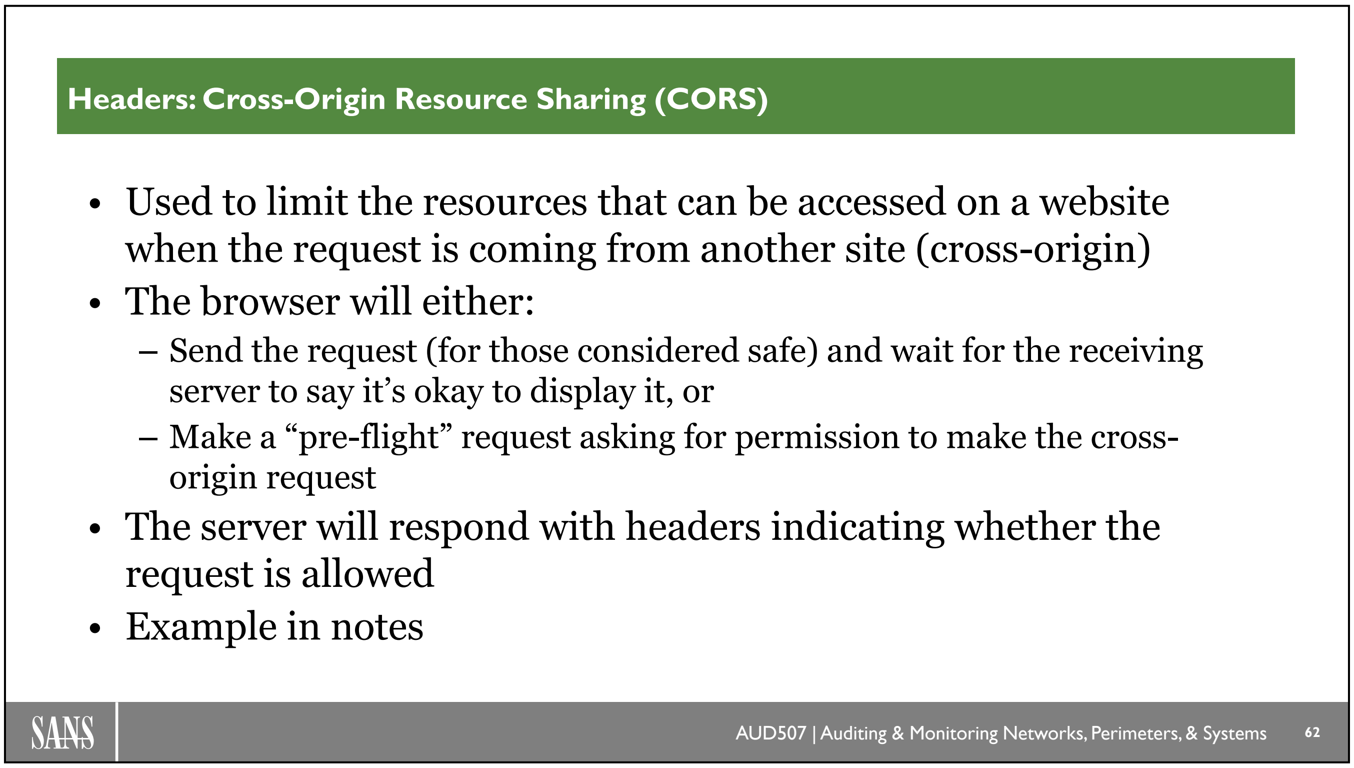
If, however, I'm on https://evilwebsite.com and my browser is about to follow a link to https://victim.com, the browser will treat this as cross-origin and may not allow certain things, like passing session cookies, to happen.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Headers: Cross-Origin Resource Sharing (CORS)

- Used to limit the resources that can be accessed on a website when the request is coming from another site (cross-origin)
- The browser will either:
  - Send the request (for those considered safe) and wait for the receiving server to say it's okay to display it, or
  - Make a "pre-flight" request asking for permission to make the cross-origin request
- The server will respond with headers indicating whether the request is allowed
- Example in notes

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 62

Imagine that your browser is rendering a page from www.goodsite.com and is about to load a cross-origin resource (like a GET request for the contents of a webpage into an iframe). The browser will send the request, including an "origin" header to let the server know what site generated the request:

```
GET /resource HTTP/1.1
Origin: http://www.goodsite.com
Host: api.webapp.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla
...
```

August 10, 2021

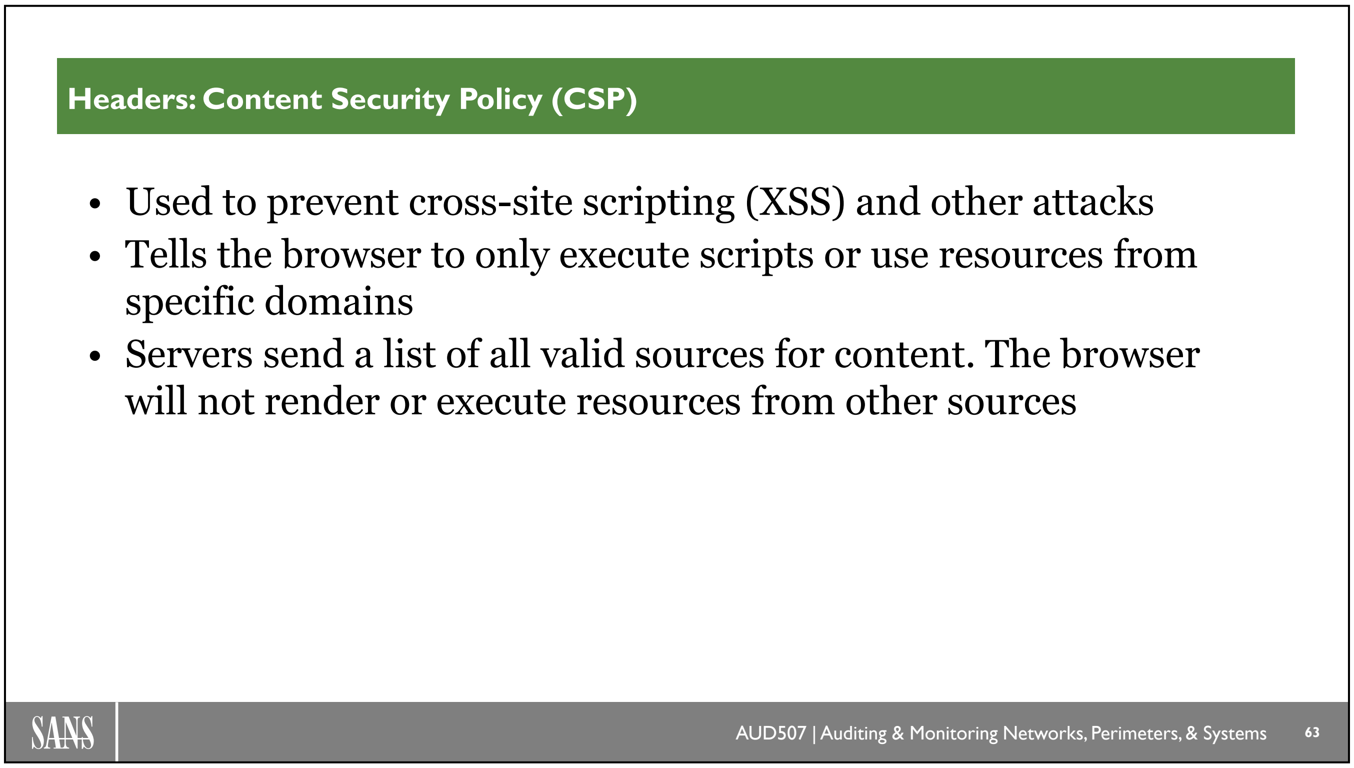
If the server chooses to allow the response to be displayed by the browser, it will add a header to its response to indicate that the access is allowed:

```
…
Access-Control-Allow-Origin: http://www.goodsite.com
Access-Control-Allow-Credentials: true
Content-Type: text/html; charset=utf-8
```

If the request is not allowed, it will not include the headers.

August 10, 2021

© SANS Institute 2021



Headers: Content Security Policy (CSP)

- Used to prevent cross-site scripting (XSS) and other attacks
- Tells the browser to only execute scripts or use resources from specific domains
- Servers send a list of all valid sources for content. The browser will not render or execute resources from other sources

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 63

The CSP header is used to limit which sources the browser will trust for things like JavaScript. The simplest CSP would allow only resources from this site to be used:

```
Content-Security-Policy: default-src 'self’
```

This policy can be much more complex for sites that source scripts and other resources from multiple domains. Here’s a portion of the CSP that was in use on twitter on the day I wrote this slide.

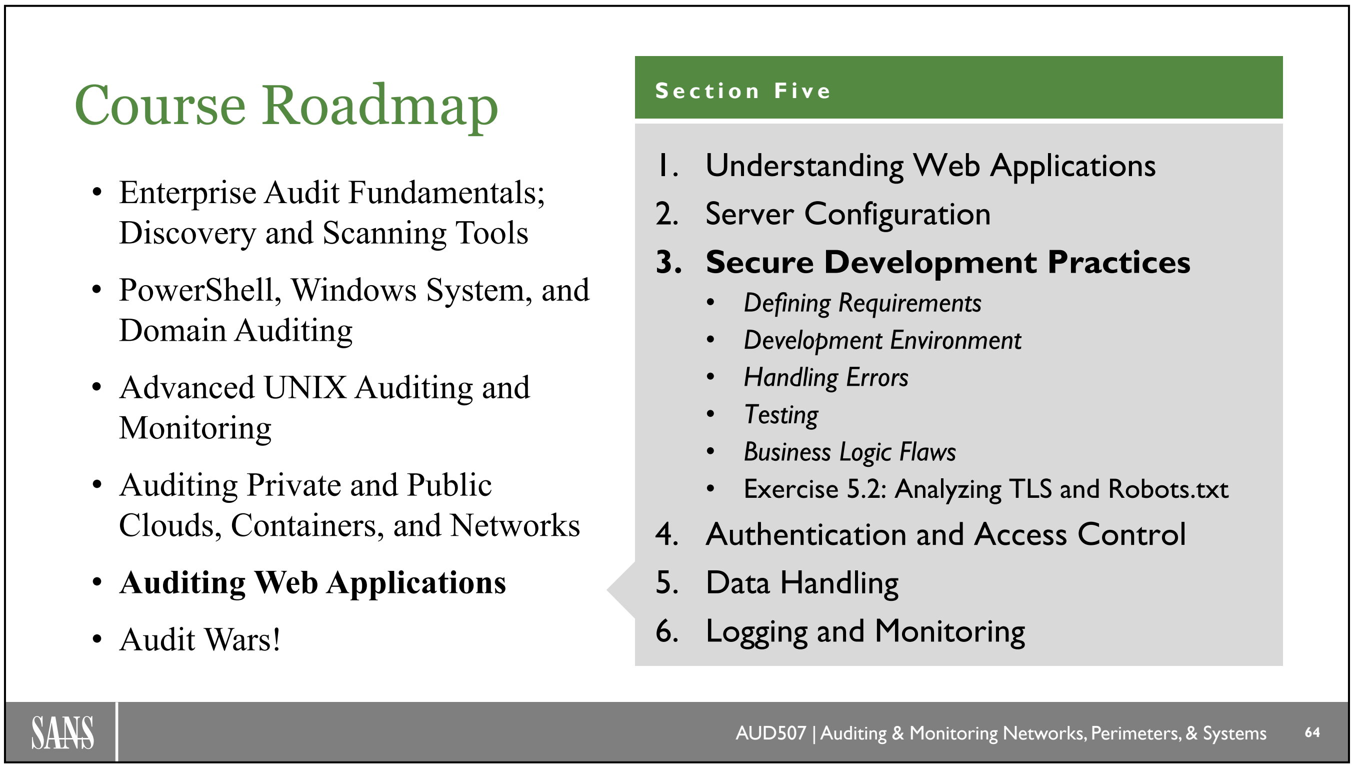
content-security-policy: connect-src 'self' blob: https://*.giphy.com https://*.pscp.tv https://*.video.pscp.tv https://*.twimg.com https://api.twitter.com https://caps.twitter.com https://media.riffsy.com https://pay.twitter.com https://sentry.io https://ton.twitter.com https://twitter.com https://upload.twitter.com https://www.google-analytics.com https://app.link https://api2.branch.io https://bnc.lt https://dwo3ckksxlb0v.cloudfront.net ; default-src 'self'; form-action 'self' https://twitter.com https://*.twitter.com; font-src 'self' https://*.twimg.com; frame-src 'self' https://twitter.com https://mobile.twitter.com https://pay.twitter.com https://cards-frame.twitter.com ; img-src 'self' blob: data: https://*.cdn.twitter.com https://ton.twitter.com https://*.twimg.com https://www.google-analytics.com https://www.periscope.tv https://www.pscp.tv https://media.riffsy.com https://*.giphy.com https://*.pscp.tv; manifest-src 'self'; media-src 'self' blob: https://twitter.com https://*.twimg.com https://*.vine.co https://*.pscp.tv https://*.video.pscp.tv https://*.giphy.com https://media.riffsy.com https://mdhdsnappytv-vh.akamaihd.net https://mpdhdsnappytv-vh.akamaihd.net https://rmpdhdsnappytv-vh.akamaihd.net https://rmmdhdsnappytv-vh.akamaihd.net https://dwo3ckksxlb0v.cloudfront.net; object-src 'none'; script-src 'self' 'unsafe-inline' https://*.twimg.com https://www.google-analytics.com https://twitter.com https://app.link 'nonce-NzhhNjU2YTMtZTdjMy00YzdkLTg3M2MtYzgzOTRjNTE3NzRh'; style-src 'self' 'unsafe-inline' https://*.twimg.com; worker-src 'self' blob:; report-uri https://twitter.com/i/csp_report?a=O5RXE%3D%3D%3D&ro=false

More information on CSP is available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

August 10, 2021

Technet24

© SANS Institute 2021



Course Roadmap

- Enterprise Audit Fundamentals; Discovery and Scanning Tools
- PowerShell, Windows System, and Domain Auditing
- Advanced UNIX Auditing and Monitoring
- Auditing Private and Public Clouds, Containers, and Networks
- **Auditing Web Applications**
- Audit Wars!

Section Five

1. Understanding Web Applications
2. Server Configuration
3. **Secure Development Practices**
   - *Defining Requirements*
   - *Development Environment*
   - *Handling Errors*
   - *Testing*
   - *Business Logic Flaws*
   - Exercise 5.2: Analyzing TLS and Robots.txt
4. Authentication and Access Control
5. Data Handling
6. Logging and Monitoring

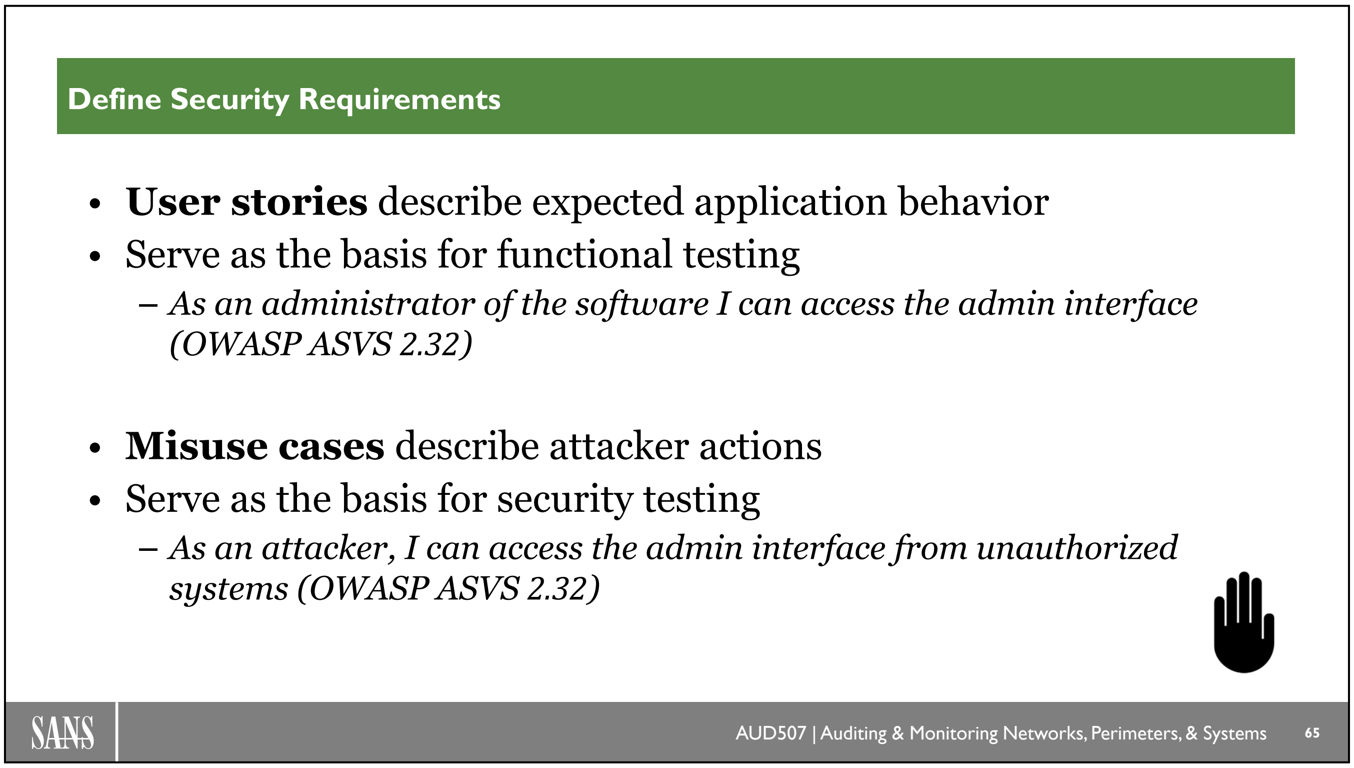
SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 64

This page intentionally left blank.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Define Security Requirements

- **User stories** describe expected application behavior
- Serve as the basis for functional testing
  - *As an administrator of the software I can access the admin interface (OWASP ASVS 2.32)*

- **Misuse cases** describe attacker actions
- Serve as the basis for security testing
  - *As an attacker, I can access the admin interface from unauthorized systems (OWASP ASVS 2.32)*

SANS

AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 65

OWASP recommends in Proactive Control 1 that developers use user stories and misuse cases to define security requirements for the application. A user story is a simple statement of a function that will be required by a user of the application. An example would be, "as a user of the application, I can create a username and password during enrollment."

A misuse case is a special user story that describes an abuse case from the perspective of the attacker. An example is, "as an attacker I can enumerate the usernames of existing users by attempting to enroll multiple usernames."

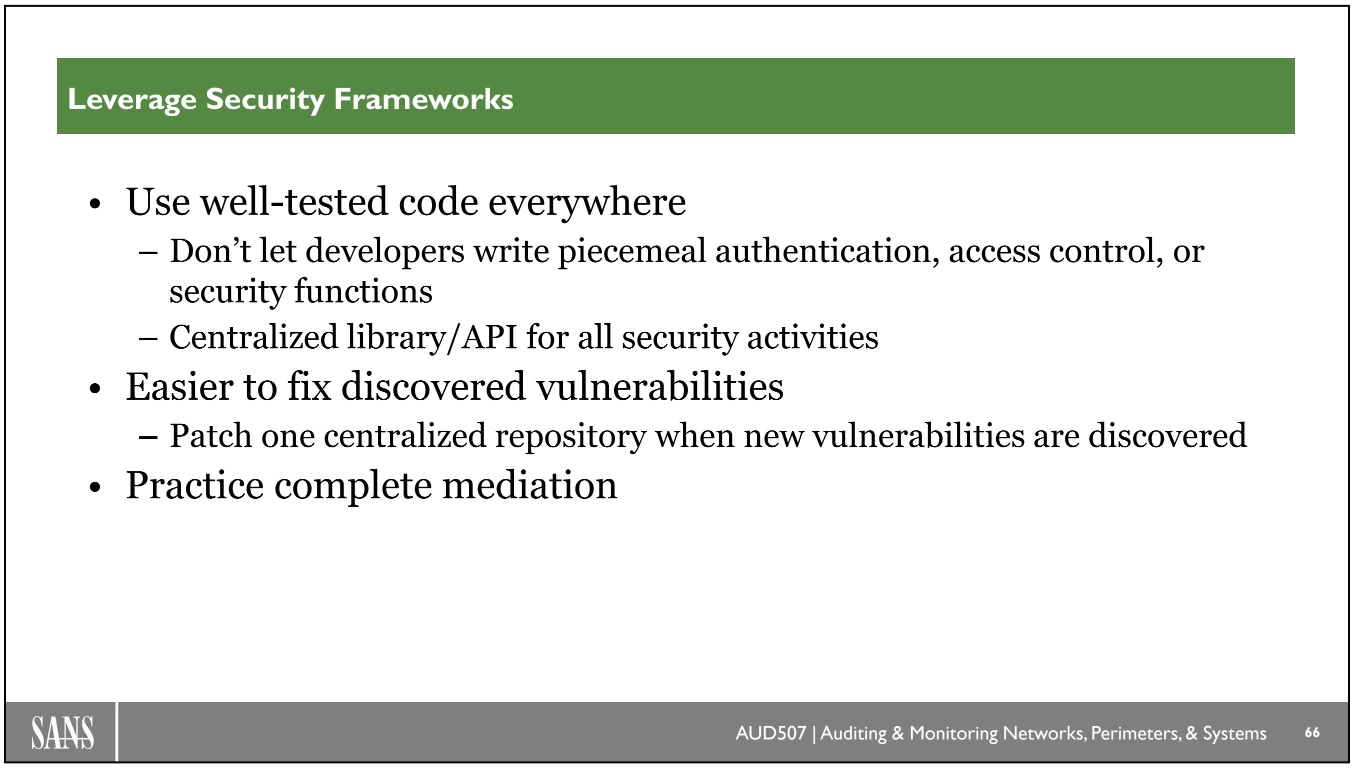
These use/misuse cases then become the foundations of the functional and security testing which will be performed against the application.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Leverage Security Frameworks

- Use well-tested code everywhere
  - Don't let developers write piecemeal authentication, access control, or security functions
  - Centralized library/API for all security activities
- Easier to fix discovered vulnerabilities
  - Patch one centralized repository when new vulnerabilities are discovered
- Practice complete mediation

AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 66

Many enterprises suffer from "summer intern syndrome," in which their web applications are a hodgepodge of components written by (often temporary) developers over the years. One developer handled authentication one way, the next implemented something new from scratch. Imagine that after a few years of this, an authentication flaw is found in part of the web application. Finding whether the flaw exists elsewhere will be very difficult because of the inconsistent code base in use.

We should recommend to our organizations that they acquire or develop a single, centralized programming function or application programming interface (API) for all security functions and that they require all of their programmers to use those functions – never write their own. By enforcing centralized code repositories for sensitive functions, it becomes much easier for the organization to patch vulnerabilities across the enterprise as soon as they are discovered.

August 10, 2021

August 10, 2021

© SANS Institute 2021

Complete Mediation

- Consider a single point of entry:
  - "Myapp.com/index.php"
  - Everything goes through here
  - Nothing else is accessible on the site without passing through here
  - Reuse of reliable code

Consider requiring complete mediation for your web application. The principle of complete mediation is that there is only one way in or out in a given architecture. For a web application you might think of this as meaning that all interaction with the web application goes through a single point of entry rather than the user requesting many different files and directories from the server.

Within a web application, having a "gatekeeper" or "router" function through which every other function is accessed provides a single point where all general authentication and authorization activities occur. It is always possible to add additional authorization checks deeper inside of the application, but this gatekeeper acts as a first line of defense across the entirety of the application. We do not have to worry that the developer has forgotten to use the security library when he adds new features to the application. This would include API calls over AJAX. These may require some specific examination since it is common to find that the security model has been bypassed, sometimes completely, for API calls.

An example of how to do this is to create the application in such a way that it essentially acts as a server, handling requests and distributing content based on validated session credentials for the clients. If this mediation point were in the index.php page, that would mean that virtually no other content in the application is accessible *without* making the request through index.php. Typically, some exceptions are considered acceptable in this framework – images and JavaScript libraries, for instance.

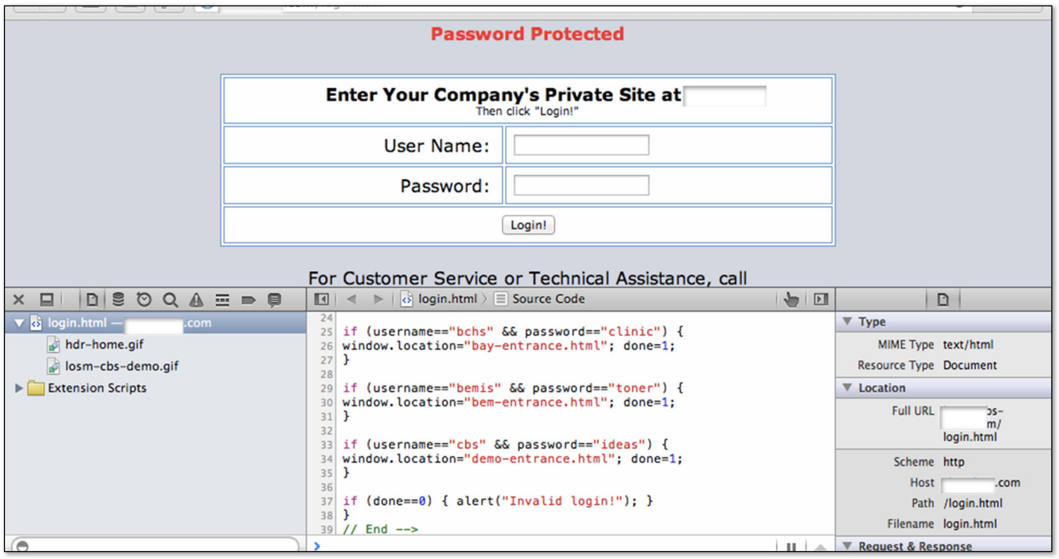
Requiring that we have a single point of access for all requests, puts us in a position to create a single set of robust functions to handle all client interaction and reuse those for all requests. This makes debugging and later security fixing much easier, though it does tend to require a fair amount of planning up front to get it right.

August 10, 2021

August 10, 2021
Technet24

© SANS Institute 2021

Client-Side Authentication?



```
if (username=="bchs" && password=="clinic") {
window.location="bay-entrance.html"; done=1;
}

if (username=="bemis" && password=="toner") {
window.location="bem-entrance.html"; done=1;
}

if (username=="cbs" && password=="ideas") {
window.location="demo-entrance.html"; done=1;
}

if (done==0) { alert("Invalid login!"); }
}
// End -->
```

Type	
MIME Type	text/html
Resource Type	Document
Location	
Full URL	[redacted]os-m/login.html
Scheme	http
Host	[redacted].com
Path	/login.html
Filename	login.html
Request & Response	

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 68

Here is an example of a REALLY bad way to handle authentication. This developer either does not know how to write the appropriate server-side authentication code, or they are unaware that browsers have a "view source" function.

Looking at the webpage, you can see that the user is prompted to log in with a username and password. Look carefully at the source code visible in the bottom one-half of the screenshot. You can see that the JavaScript is checking to see if certain user ID and password combinations have been entered. If they have, the user is redirected to the appropriate webpage. If the user does not enter one of these combinations, an alert box opens saying, "Invalid login!"

This is just one example of why we want our best developers writing our security code, and why we need rules that force other developers to use only that code.

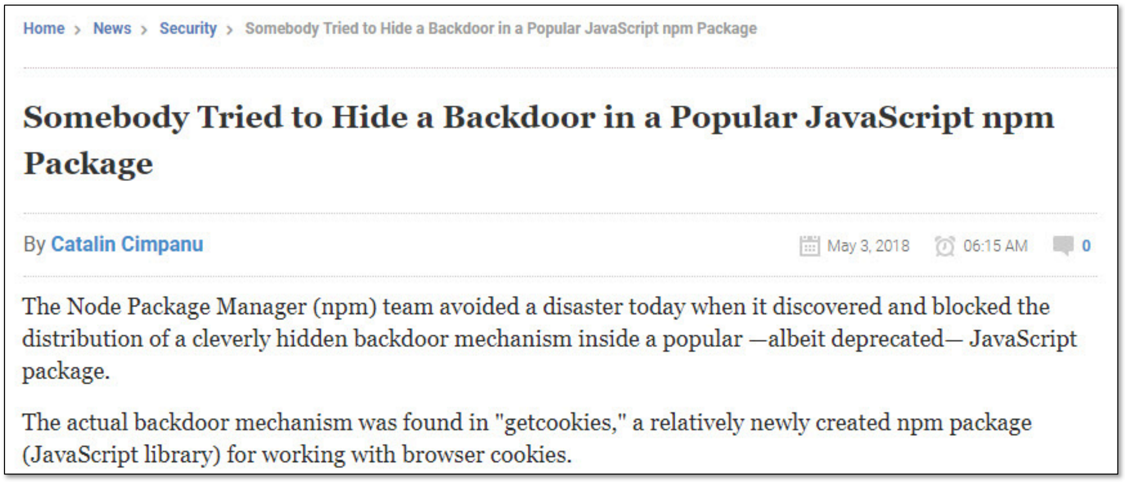
August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Keep Your Libraries Up To Date



Home > News > Security > Somebody Tried to Hide a Backdoor in a Popular JavaScript npm Package

Somebody Tried to Hide a Backdoor in a Popular JavaScript npm Package

By Catalin Cimpanu May 3, 2018 06:15 AM 0

The Node Package Manager (npm) team avoided a disaster today when it discovered and blocked the distribution of a cleverly hidden backdoor mechanism inside a popular —albeit deprecated— JavaScript package.

The actual backdoor mechanism was found in "getcookies," a relatively newly created npm package (JavaScript library) for working with browser cookies.

Using components with known vulnerabilities is one of the OWASP Top Ten as of the 2017 edition. This issue is often caused by developers importing libraries and other packages into a project, or perhaps settling on a specific version of an application framework and failing to update the libraries or framework as development continues.

Rarely is there a conscious decision to use flawed code. Instead, the vulnerability was likely unknown when the choice was first made, but later issues were discovered (like the extreme issue described in the article in the slide!). Since developers aren't focused on following security patches and upgrades to packages underlying code, however, it is likely that the developers are completely unaware that the code that they are relying on is now known to be flawed. Worse, as more and more code is written without keeping all libraries and the framework up to date, the code is fast becoming dependent on the flawed code. In other words, the more time that passes without updating the libraries or framework, the less likely that we can upgrade easily; our code will require major revisions to work with the adjustments to the framework or libraries that have been updated!

August 10, 2021

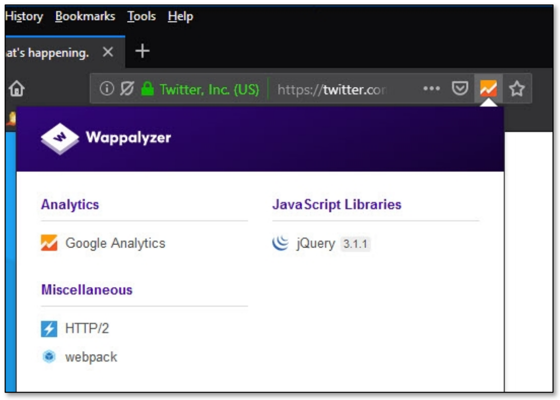
August 10, 2021

Technet24

© SANS Institute 2021

Finding Plugins and Frameworks: Wappalyzer

- Used to find plugins and programming frameworks used on a website
- Available as browser plugin for Chrome and Firefox
- Gives a quick look at potentially vulnerable components on a site



The Wappalyzer browser plugin allows the web application auditor to determine the technologies being used by a web application. It recognizes common language frameworks, content management systems, Javascript libraries, and analytics tools, and shows the version numbers of the components found. Using this plugin can greatly speed up the process of searching for out-of-date or vulnerable components used in an application.

Wappalyzer is available as a browser plugin for both Chrome and Firefox. It is open source software whose source code can be found on GitHub.

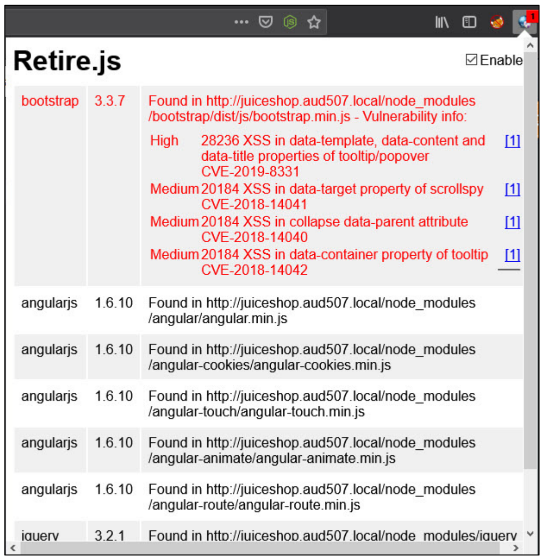
August 10, 2021

August 10, 2021

© SANS Institute 2021

Retire.JS

- Retire.JS finds outdated JavaScript libraries used by a site
- Installs as:
  - Browser plugin
  - Burp or ZAP plugin
  - Command-line tool
  - Grunt plugin for DevOps teams



Retire.js ☑ Enable

bootstrap	3.3.7	Found in http://juiceshop.aud507.local/node_modules /bootstrap/dist/js/bootstrap.min.js - Vulnerability info: High 28236 XSS in data-template, data-content and data-title properties of tooltip/popover CVE-2019-8331 [1]; Medium 20184 XSS in data-target property of scrollspy CVE-2018-14041 [1]; Medium 20184 XSS in collapse data-parent attribute CVE-2018-14040 [1]; Medium 20184 XSS in data-container property of tooltip CVE-2018-14042 [1]
angularjs	1.6.10	Found in http://juiceshop.aud507.local/node_modules /angular/angular.min.js
angularjs	1.6.10	Found in http://juiceshop.aud507.local/node_modules /angular-cookies/angular-cookies.min.js
angularjs	1.6.10	Found in http://juiceshop.aud507.local/node_modules /angular-touch/angular-touch.min.js
angularjs	1.6.10	Found in http://juiceshop.aud507.local/node_modules /angular-animate/angular-animate.min.js
angularjs	1.6.10	Found in http://juiceshop.aud507.local/node_modules /angular-route/angular-route.min.js
jquery	3.2.1	Found in http://juiceshop.aud507.local/node_modules/jquery

The Retire.JS tool is a scanner built to discover vulnerable JavaScript libraries being used by a web application. One of the great things about Retire.JS is that it can be used in several different ways:

- As a browser plugin for Firefox or Chrome
- As a plugin in attack proxies like Burp and ZAP
- As a stand-alone command-line tool
- Maybe the most useful: As a plugin for the Grunt integration tool used by many DevOps teams for automating code check-in and deployment

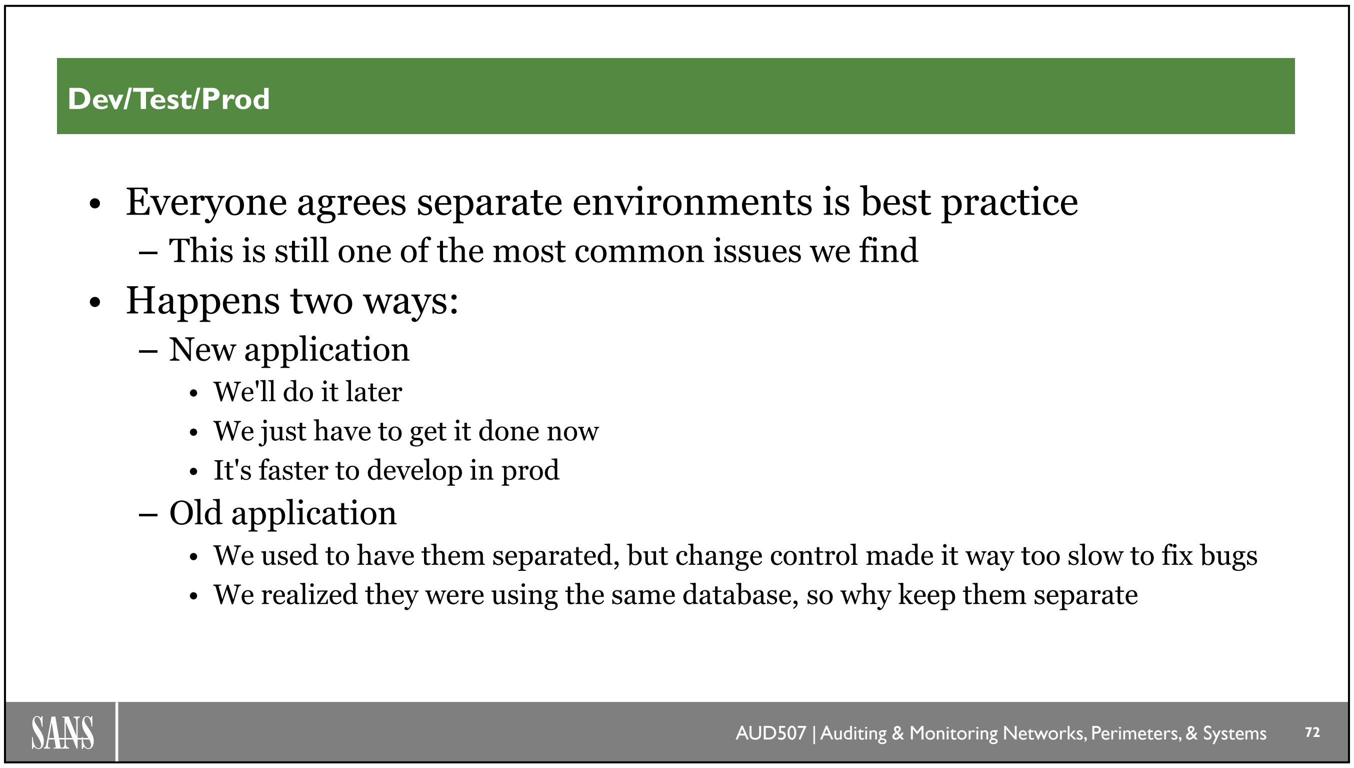
We have the Retire.JS plugin installed in Firefox on the AUD507 student VM.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Dev/Test/Prod

- Everyone agrees separate environments is best practice
  - This is still one of the most common issues we find
- Happens two ways:
  - New application
    - We'll do it later
    - We just have to get it done now
    - It's faster to develop in prod
  - Old application
    - We used to have them separated, but change control made it way too slow to fix bugs
    - We realized they were using the same database, so why keep them separate

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 72

One of the most common issues that we find is that the organization has failed to enforce the requirement for separate development, test, and production environments. This is exceptionally dangerous.

Not only does this mean that largely untested code running in production risks exposing or damaging production data, but it also means that this same largely untested code is being exposed to potential attackers with little or, more likely, no real security testing. Bear in mind the point that we made during our discussion of automated testing tools. Even if we are using an automated testing tool, it is unlikely to find anything except for the most common and egregious coding errors. They will *never* find logic errors in our code that allow users to do things that they ought not, nor will it be able to detect information disclosures where too much data is being sent to the client. (We will see some examples of this when we begin examining single-page applications more closely)

If everyone agrees that this is terrible practice, how does it happen so often? In our experience, there are two paths that lead to this situation. One is the new application that is being developed under a tight deadline. The team claims that having separate environments will just slow them down, so they explain that they will "do it later," after the application is delivered. Rarely will this change happen.

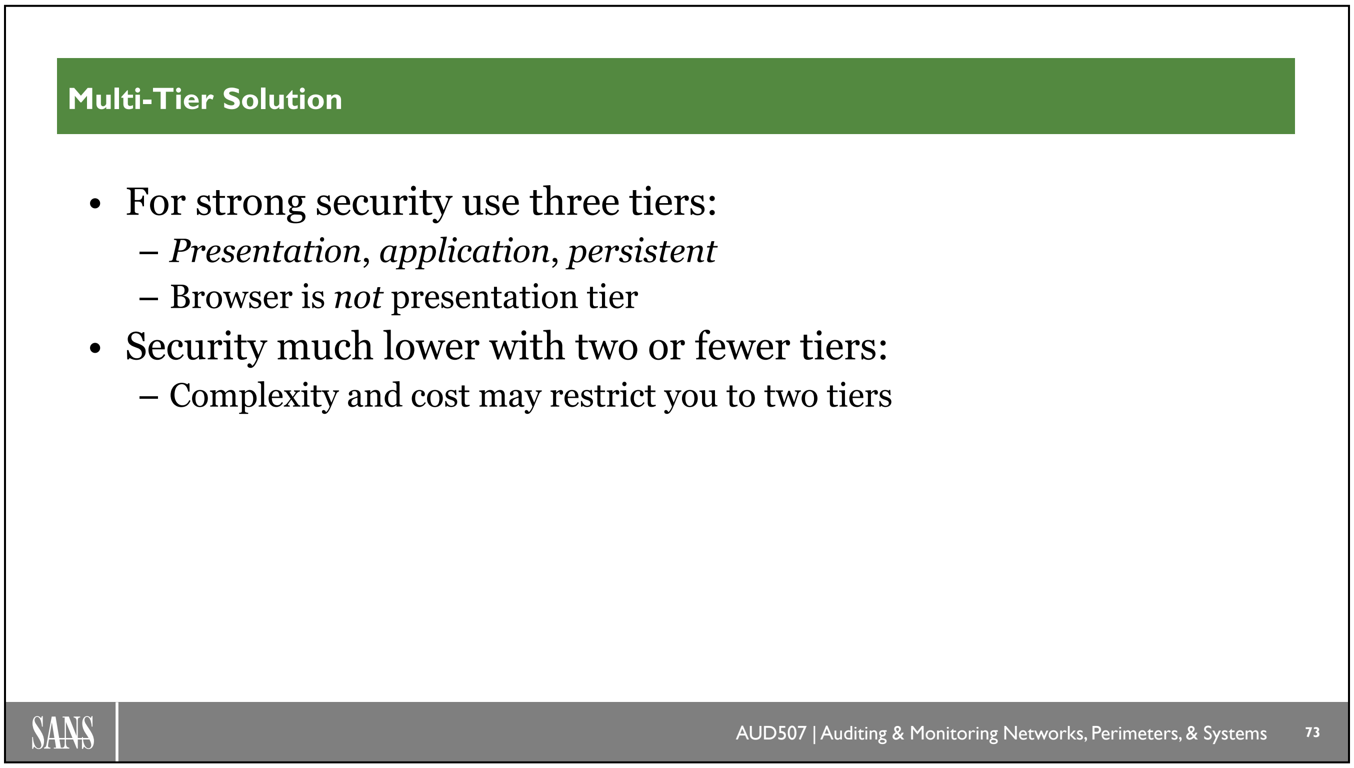
The second path is the older application that did, in fact, have separate environments at some point in the past. Over time, administrators found it challenging to keep the three synchronized because developers were permitted to make undocumented changes to Dev, leading to situations where approved code would fail when moved to production. You may also hear that the environments were collapsed because the change control process was too onerous, leading to long delays for changes to make it into production. Especially if the latter is the case, it is often an indicator that the quality of code being produced is low, leading to frequent and rapid bug fixes to be deployed to production.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Multi-Tier Solution

- For strong security use three tiers:
  - *Presentation*, *application*, *persistent*
  - Browser is *not* presentation tier
- Security much lower with two or fewer tiers:
  - Complexity and cost may restrict you to two tiers

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 73

Your organization should consider implementing an application as a three-tier solution. The idea behind a three-tier solution is that it provides for wonderful scalability prospects in addition to tightly controlling any sensitive information that might be a part of the application or stored behind the application. The tiers are typically defined as:

- **Presentation:** The presentation tier is usually a web server that hands static pages back to the client and essentially acts as a proxy of sorts for any active or dynamic content. A portal is a good example of a presentation tier.
- **Application:** The application tier handles all dynamic content and interaction with the persistent tier, acting as a sort of dynamic gateway to generate dynamic content from the persistent tier, serving that content to the presentation tier.
- **Persistent:** The persistent tier is quite simply a database. This tier handles all long-term storage and retrieval of data that will eventually be delivered through our frontend.

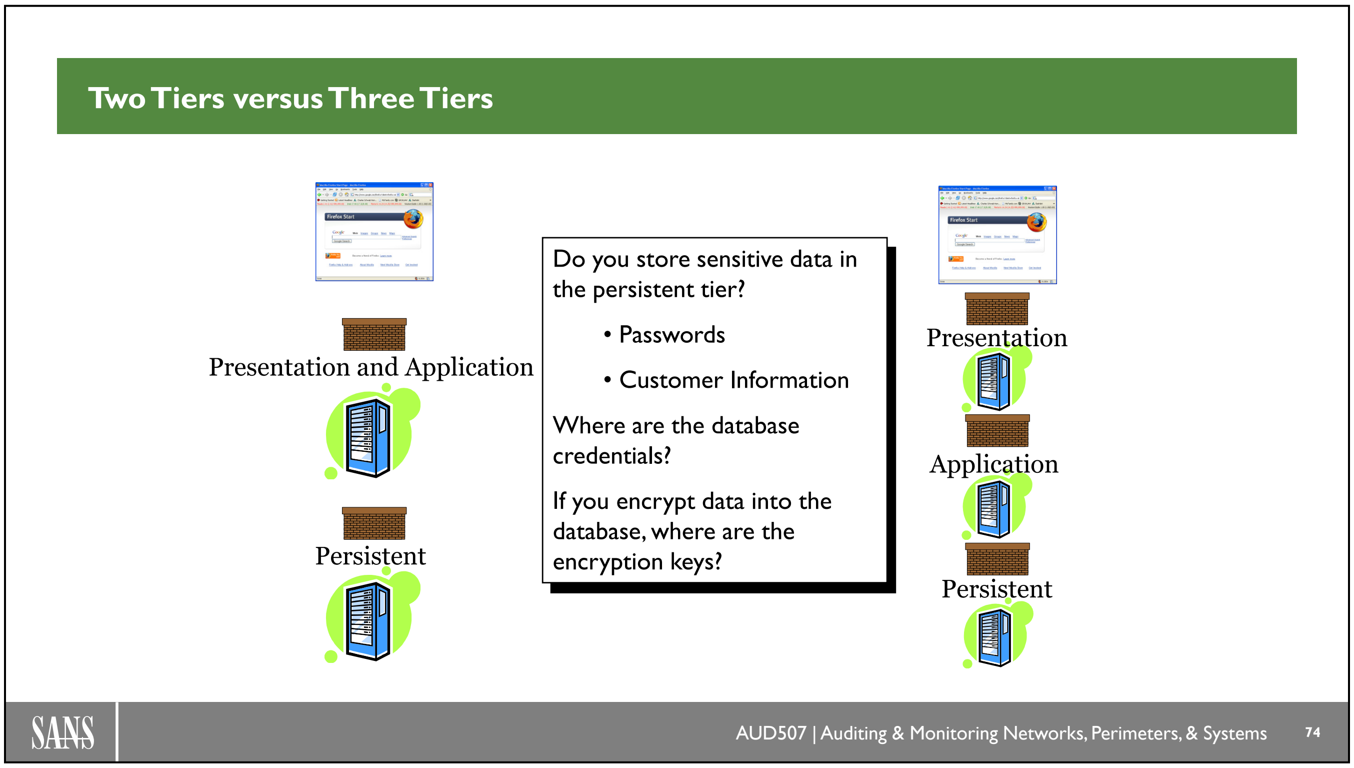
We must acknowledge that although a three-tier application is best practice, many find the development, deployment, and maintenance costs too high for all applications. For this reason, we suggest that you decide how many tiers to use based on the security requirements for the data that you are handling.

August 10, 2021

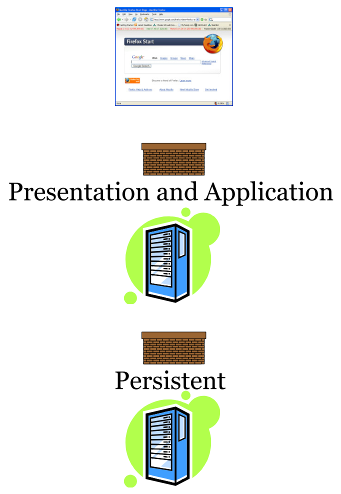
August 10, 2021

Technet24

© SANS Institute 2021



Two Tiers versus Three Tiers

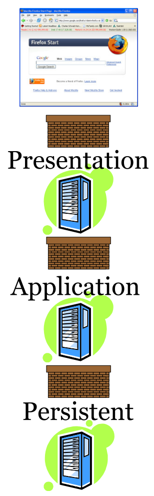


Do you store sensitive data in the persistent tier?

- Passwords
- Customer Information

Where are the database credentials?

If you encrypt data into the database, where are the encryption keys?



For a moment, let's put the two-tier and three-tier architectures side by side so that we can examine the strengths and weaknesses. For a three-tier environment, the obvious problems reside in complexity and cost. Clearly, developing an application that leverages all three tiers will be time-consuming and the hardware requirements will be costly. Even so, if scalability and security are our major concerns, this is definitely the way to go.

Look at the two-tier diagram and consider the questions in the center of the slide. In a three-tier environment, the presentation tier is never permitted to talk directly to the persistent tier. Instead, all requests must be made through the application tier. Similarly, the application tier can never communicate directly with the internet. Instead, everything that the application tier sends must be processed through the presentation tier.

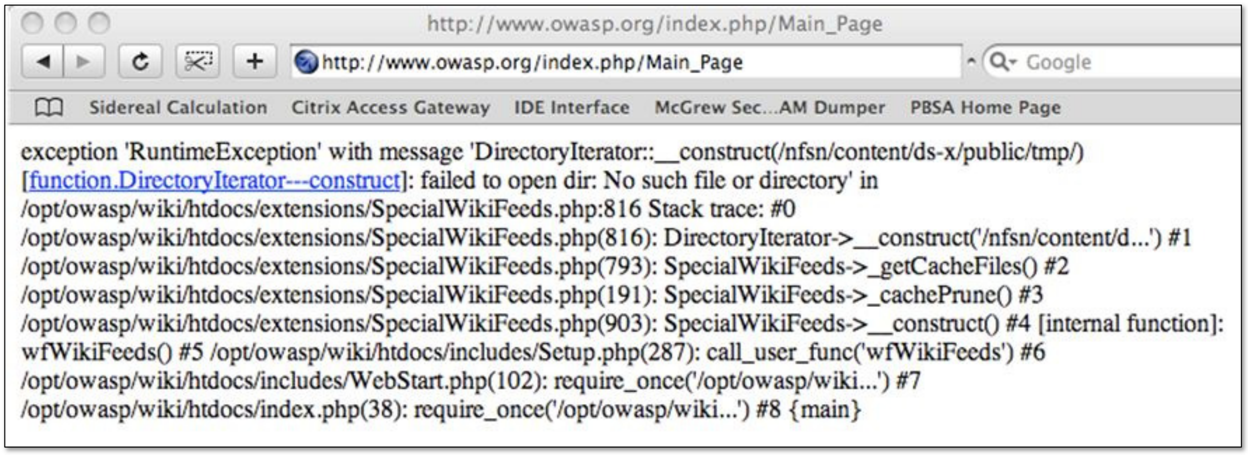
Whatever the case, the credentials that are used to access the persistent tier must reside in the application tier. If the application and presentation tier are joined, then compromising either of those tiers compromises the other. Because the credentials for the persistent tier are there, too, a compromise of either tier actually results in the compromise of all three tiers, since it gives the attacker credentials that can be used to extract all of the data from the database. In some requests, we have only a single tier. This is not true in a (well-implemented) three-tier system.

August 10, 2021

August 10, 2021

© SANS Institute 2021

Error Handling: If OWASP Can Make This Mistake…



```
exception 'RuntimeException' with message 'DirectoryIterator::__construct(/nfsn/content/ds-x/public/tmp/)
[function.DirectoryIterator---construct]: failed to open dir: No such file or directory' in
/opt/owasp/wiki/htdocs/extensions/SpecialWikiFeeds.php:816 Stack trace: #0
/opt/owasp/wiki/htdocs/extensions/SpecialWikiFeeds.php(816): DirectoryIterator->__construct('/nfsn/content/d...') #1
/opt/owasp/wiki/htdocs/extensions/SpecialWikiFeeds.php(793): SpecialWikiFeeds->_getCacheFiles() #2
/opt/owasp/wiki/htdocs/extensions/SpecialWikiFeeds.php(191): SpecialWikiFeeds->_cachePrune() #3
/opt/owasp/wiki/htdocs/extensions/SpecialWikiFeeds.php(903): SpecialWikiFeeds->__construct() #4 [internal function]:
wfWikiFeeds() #5 /opt/owasp/wiki/htdocs/includes/Setup.php(287): call_user_func('wfWikiFeeds') #6
/opt/owasp/wiki/htdocs/includes/WebStart.php(102): require_once('/opt/owasp/wiki...') #7
/opt/owasp/wiki/htdocs/index.php(38): require_once('/opt/owasp/wiki...') #8 {main}
```

Here is an example of a programmer failing to properly handle errors. This issue is disturbingly common to find. The root cause of this problem is that programmers tend to check only for error conditions that they expect can occur. Users, however, are extremely good at trying things that the programmer never envisioned.

Every major language used for web application development today supports the ability to catch even unexpected errors. In this case, we're looking at a host run by OWASP. The page accessed is a PHP-driven page. This type of error should *never* be displayed to a user. Not only is this disclosing information that we just don't need to have, but it also affects consumer confidence.

Realize that we are not including this screenshot to cast OWASP in a bad light! To the contrary, this is a group of smart and skilled people who are doing good work in the web application security space! Instead, consider this: **If these folks could make this kind of mistake, what are the chances that *our* programmers are making these mistakes?**

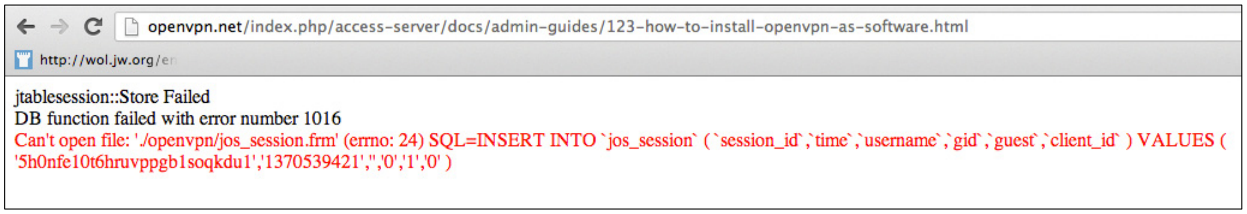
From this error alone, we can derive a lot of useful details for an attacker. For instance, at first glance we can see that some kind of access to a directory failed. Looking at the details of the error message, it leads us to the conclusion that we are looking, most likely, at a farm of frontend web servers that are used to deliver content from a backend SAN operating over NFS. It appears that the NFS server is unexpectedly down, but the programmer simply assumes that the server will always exist.

August 10, 2021

Technet24

© SANS Institute 2021

A More Frightening Error Message


openvpn.net/index.php/access-server/docs/admin-guides/123-how-to-install-openvpn-as-software.html
http://wol.jw.org/e

```
jtablesession::Store Failed
DB function failed with error number 1016
Can't open file: './openvpn/jos_session.frm' (errno: 24) SQL=INSERT INTO `jos_session` ( `session_id`,`time`,`username`,`gid`,`guest`,`client_id` ) VALUES (
'5h0nfe10t6hruvppgb1soqkdu1','1370539421','','0','1','0' )
```

Here's another frightening example of an error message that says far too much. OpenVPN are the folks who create and distribute both the free and commercial OpenVPN products that are widely used for VPNs in small- and even medium-sized businesses. As you can see, this example actually gives us insight into the actual database schema. More than this, we may identify values that we can control as a user that end up in the SQL statement, potentially allowing us to craft a SQL injection attack.

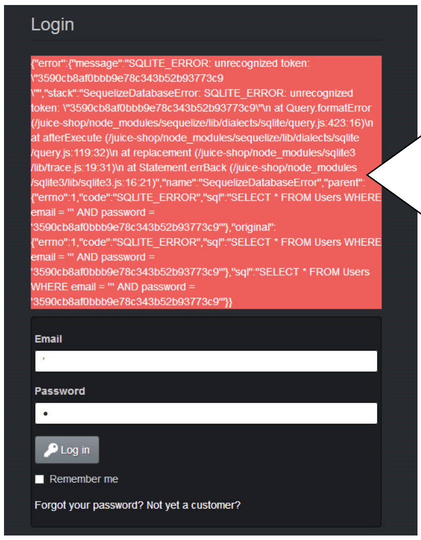
Programmers must be sure to capture every possible error condition that can happen, even if the developer cannot imagine a way that the error could occur. While this might sound as though it is difficult to accomplish, in fact, it isn't. Every major language and framework that is used for enterprise web application development today provides the programmer with the ability to capture errors that were *not* expected to occur. This makes the display in the slide above absolutely inexcusable.

August 10, 2021

August 10, 2021

© SANS Institute 2021

Some Errors Display in the Browser…



```
{"error":{"message":"SQLITE_ERROR: unrecognized token:
\"3590cb8af0bbb9e78c343b52b93773c9
\"","stack":"SequelizeDatabaseError: SQLITE_ERROR: unrecognized
token: \"3590cb8af0bbb9e78c343b52b93773c9\"\n at Query.formatError
(/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:423:16)\n
at afterExecute (/juice-shop/node_modules/sequelize/lib/dialects/sqlite
/query.js:119:32)\n at replacement (/juice-shop/node_modules/sqlite3
/lib/trace.js:19:31)\n at Statement.errBack (/juice-shop/node_modules
/sqlite3/lib/sqlite3.js:16:21)","name":"SequelizeDatabaseError","parent":
{"errno":1,"code":"SQLITE_ERROR","sql":"SELECT * FROM Users WHERE
email = ''' AND password =
'3590cb8af0bbb9e78c343b52b93773c9'"},"original":
{"errno":1,"code":"SQLITE_ERROR","sql":"SELECT * FROM Users WHERE
email = ''' AND password =
'3590cb8af0bbb9e78c343b52b93773c9'"},"sql":"SELECT * FROM Users
WHERE email = ''' AND password =
'3590cb8af0bbb9e78c343b52b93773c9'"}}
```

```
{"error"…:{"errno":1,"code":"SQLITE_ERROR"
,"sql":"SELECT * FROM Users WHERE email =
''' AND password =
'3590cb8af0bbb9e78c343b52b93773c9}}
```

Here's an example of another error message being displayed in the browser. This error is part of the Juice Shop application we're using in our labs. In this case, the error is displayed directly to the user in the webpage content. In other cases, finding the error may require just a bit more work. See the example on the next slide for another error from Juice Shop.

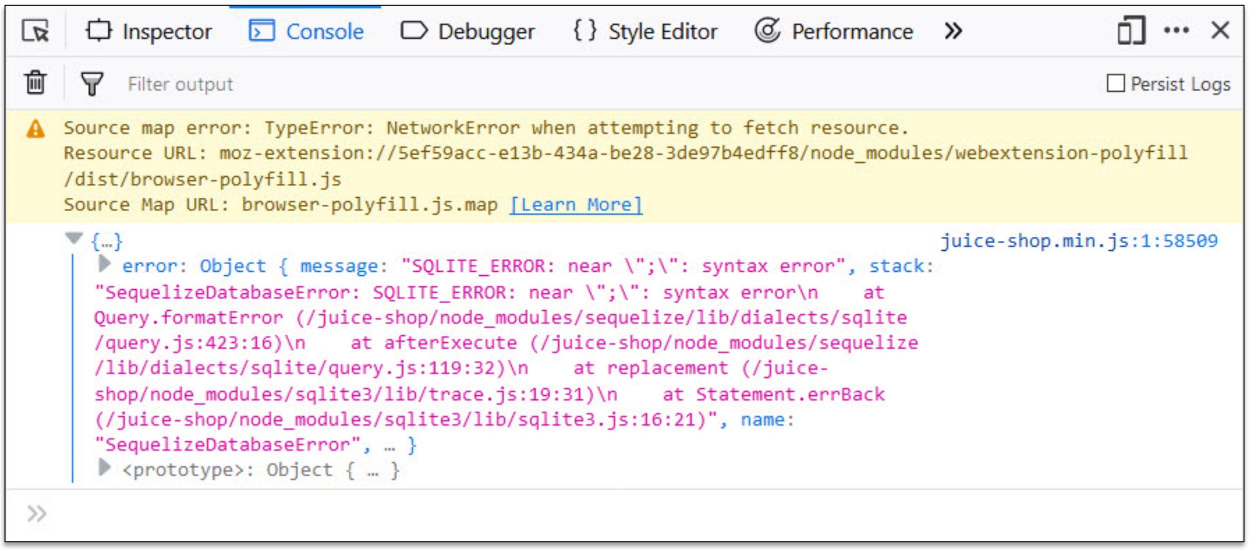
August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Errors in Single-Page App – JavaScript Console



```
Source map error: TypeError: NetworkError when attempting to fetch resource.
Resource URL: moz-extension://5ef59acc-e13b-434a-be28-3de97b4edff8/node_modules/webextension-polyfill
/dist/browser-polyfill.js
Source Map URL: browser-polyfill.js.map [Learn More]
{…}                                                                            juice-shop.min.js:1:58509
  error: Object { message: "SQLITE_ERROR: near \";\": syntax error", stack:
  "SequelizeDatabaseError: SQLITE_ERROR: near \";\": syntax error\n    at
  Query.formatError (/juice-shop/node_modules/sequelize/lib/dialects/sqlite
  /query.js:423:16)\n    at afterExecute (/juice-shop/node_modules/sequelize
  /lib/dialects/sqlite/query.js:119:32)\n    at replacement (/juice-
  shop/node_modules/sqlite3/lib/trace.js:19:31)\n    at Statement.errBack
  (/juice-shop/node_modules/sqlite3/lib/sqlite3.js:16:21)", name:
  "SequelizeDatabaseError", … }
  <prototype>: Object { … }
```

Errors in single-page apps and other JavaScript-heavy web applications will often be sent to the JavaScript console, instead of being displayed in the rendered webpage. It's a good idea to keep the console open while testing the web application so that you can see any errors displayed in the console.

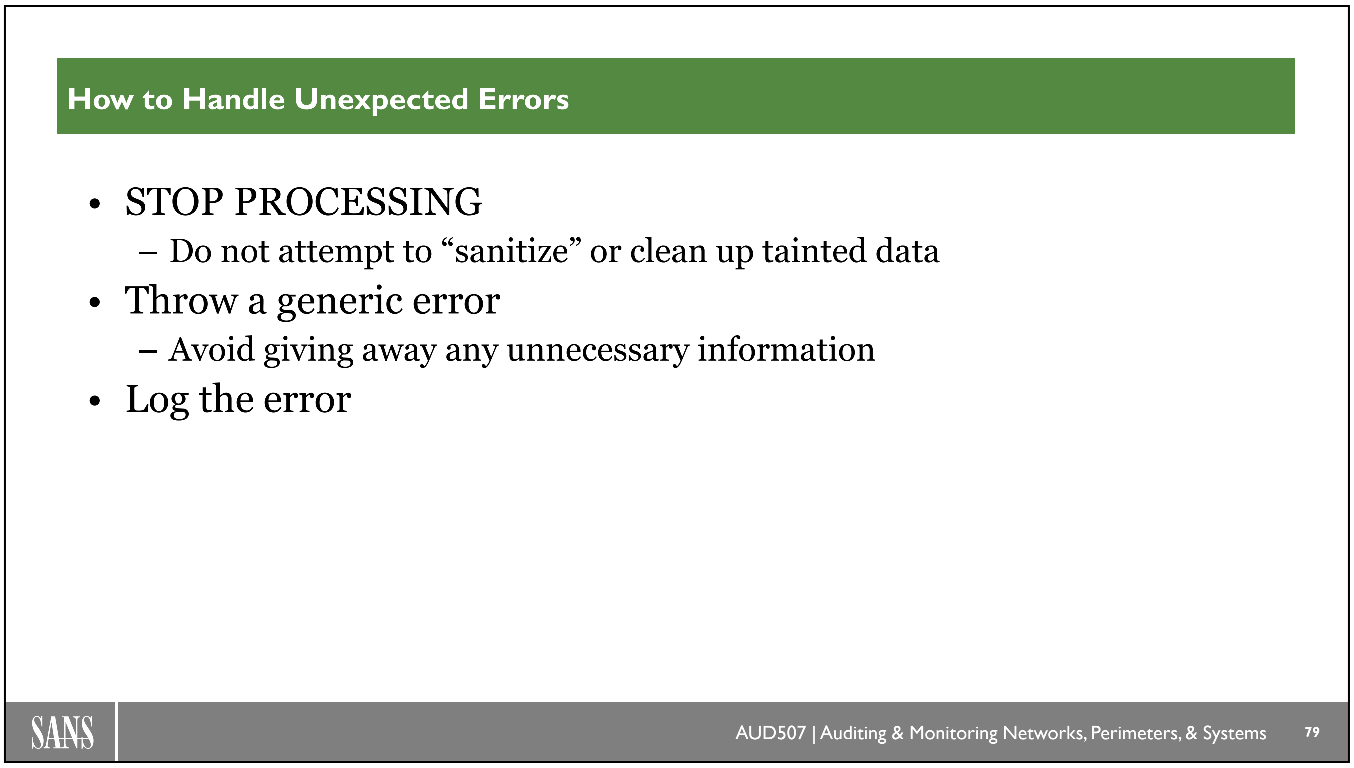
In this example, we see a SQL syntax error being output to the console from the juice-shop.min.js library used by the application. It's possible that this is an indicator of a SQL injection vulnerability that the auditor would have missed had they not been looking at the console.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



How to Handle Unexpected Errors

- STOP PROCESSING
  – Do not attempt to “sanitize” or clean up tainted data
- Throw a generic error
  – Avoid giving away any unnecessary information
- Log the error

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 79

Here’s the only good way for a developer to handle an unexpected error:

1. Stop processing data. Once an error has occurred, it is highly unlikely that the code can successfully recover without exposing the application to further attacks.

2. Throw a generic error message that gives away NO information about the nature of the error or the internal state of the application.

3. Log that the error occurred, as well as any parameters or application state data which might be useful for troubleshooting or incident handling.

Note the example on the next two slides.

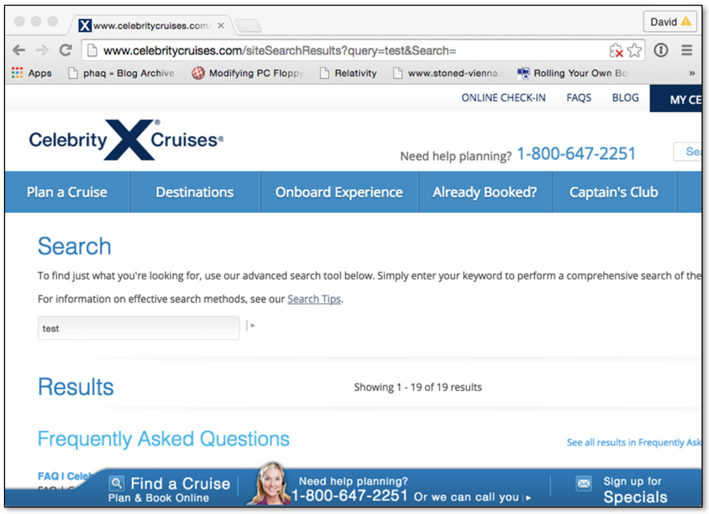
August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

What Should Happen? (1)



Search

To find just what you're looking for, use our advanced search tool below. Simply enter your keyword to perform a comprehensive search of the

For information on effective search methods, see our Search Tips.

test

Results

Showing 1 - 19 of 19 results

Frequently Asked Questions

See all results in Frequently Ask

FAQ I Cele

What should happen when an error occurs? Remember again that the main reason that errors are not handled properly is that the developer prepares only to handle errors that he expects can occur. What he *should* do as a best practice is prepare to handle absolutely any error condition, even ones that he thinks are not possible. How can he do this?

This is where the generic error-handling code mentioned on the previous page comes into play. The developer need only implement a generic error handler for uncaught exceptions.

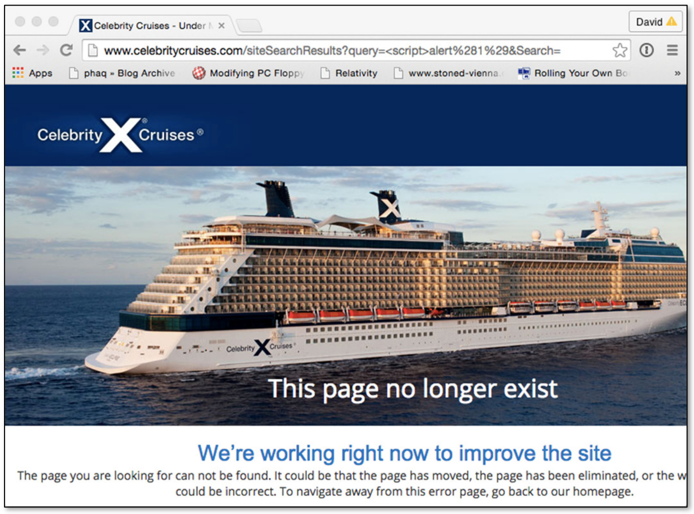
Consider the webpage pictured in the slide. Here you can see the results from a search request sent to Celebrity's website. Nothing unusual was sent, just the word "test." No error has occurred here. Compare this output with what you see on the next slide.

August 10, 2021

August 10, 2021

© SANS Institute 2021

What Should Happen? (2)



www.celebritycruises.com/siteSearchResults?query=<script>alert%281%29&Search=

This page no longer exist

We're working right now to improve the site

The page you are looking for can not be found. It could be that the page has moved, the page has been eliminated, or the w could be incorrect. To navigate away from this error page, go back to our homepage.

Here you can see what happens when the following is sent as the search term:

```
<script>alert(1)
```

Notice how different this page looks. If you search for a term that does not exist, you receive a search results page that tells you that there were no results. This page is different, however. What's going on?

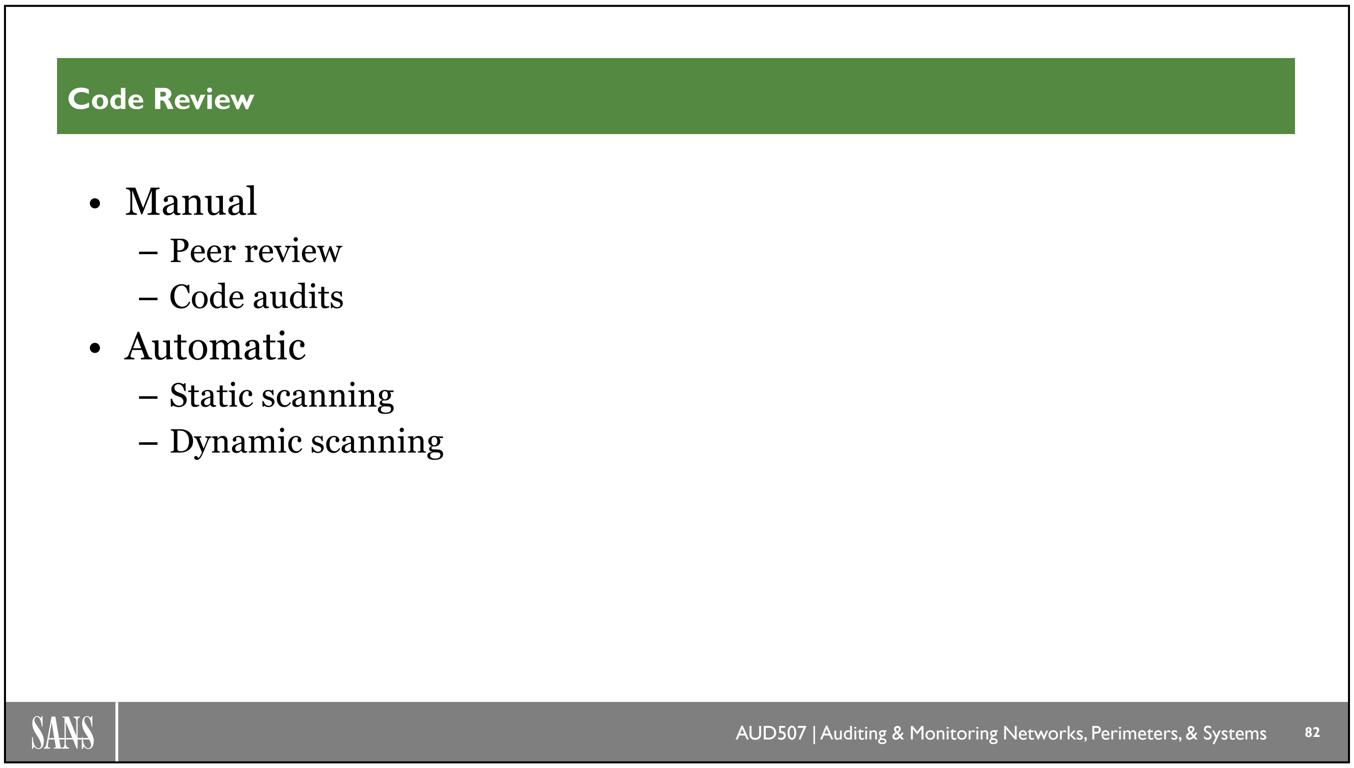
Although I would be concerned that the simple string above causes an error to occur in the application code, I am actually quite happy to find that the developer has clearly added some "catch all" error handling to his code! For some (undetermined, as far as we are concerned) reason the application encountered an error. Rather than simply crashing (and sending back a 500 Server Not Available message), the application instead returns this error page! This is what we *should* see when an unexpected error occurs!

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Code Review

- Manual
  - Peer review
  - Code audits
- Automatic
  - Static scanning
  - Dynamic scanning

The processes used for code review can be manual or automated. A combination of both approaches is used in many organizations.

Manual code review involves people looking at the code with the goal of finding security flaws. This can be done through the peer review process, in which developers meet to review each other's code for flaws, or it can be done by auditors trained to perform security reviews of code. In high-risk projects, the enterprise may use both manual techniques for best coverage.

Automated code review involves the use of scanners which seek out common flaws in code. Static analysis tools work by scanning the source code of the application, looking for flaws in the way the code is written. Dynamic tools work by testing a running copy of the application code, similar to a general-purpose vulnerability scanner. Dynamic scanning attempts to find flaws by analyzing the behavior of the running code.

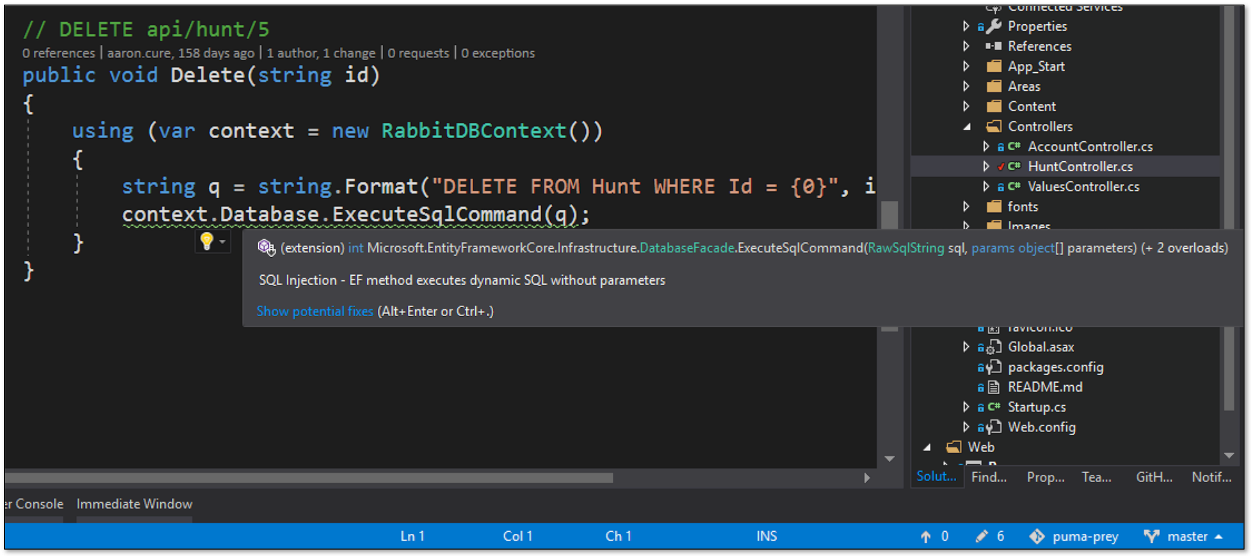
August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Static Code Analysis



```
// DELETE api/hunt/5
0 references | aaron.cure, 158 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public void Delete(string id)
{
    using (var context = new RabbitDBContext())
    {
        string q = string.Format("DELETE FROM Hunt WHERE Id = {0}", i
        context.Database.ExecuteSqlCommand(q);
    }
}
```

```
(extension) int Microsoft.EntityFrameworkCore.Infrastructure.DatabaseFacade.ExecuteSqlCommand(RawSqlString sql, params object[] parameters) (+ 2 overloads)
SQL Injection - EF method executes dynamic SQL without parameters
Show potential fixes (Alt+Enter or Ctrl+.)
```

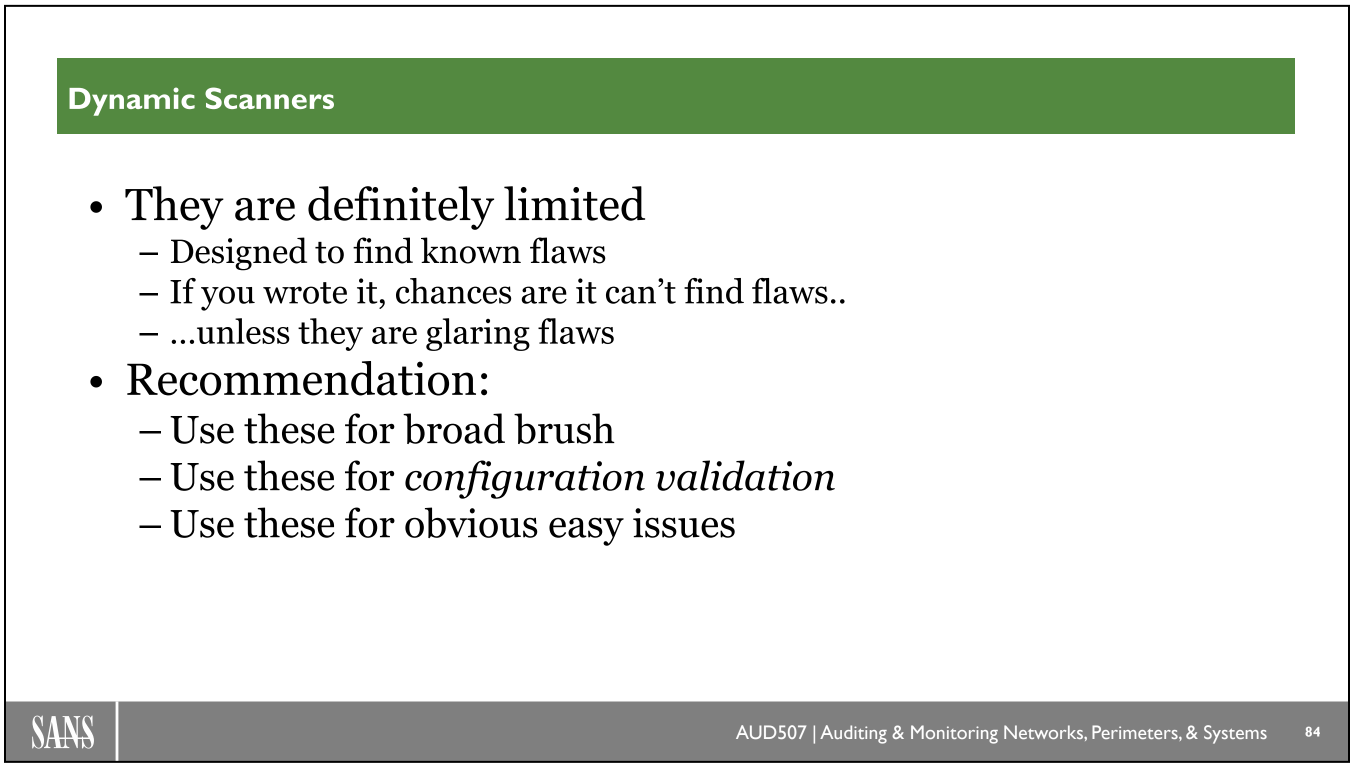
Some static analysis tools, such as Puma Scan, pictured here, analyze the code as it is being written, and warn the developer of flaws as they work. This affords the developer the chance to fix the flaws identified before committing the code to the project. Other static tools are run after the code is checked in, as part of the software build process.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Dynamic Scanners

- They are definitely limited
  - Designed to find known flaws
  - If you wrote it, chances are it can't find flaws..
  - ...unless they are glaring flaws
- Recommendation:
  - Use these for broad brush
  - Use these for *configuration validation*
  - Use these for obvious easy issues

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 84

There is a great deal of value in testing your application thoroughly, but there's a very big challenge. All of the automated scanners within this market space are designed to find known vulnerabilities in applications. They look for obvious holes like Cross-Site Scripting flaws, SQL injection flaws, and more. The trouble is that, unless it is a common flaw, these tools have a hard time figuring out that the data that just got returned from your application should not have been returned. This is not to say that these tools are useless, however! They are quite valuable, but it is important to be aware of the limitations of your tools!

Think about this in the context of the applications that your organization uses. If the applications are off-the-shelf solutions that have been purchased or licensed from a vendor, the commercial scanners will likely do a fine job of finding all known vulnerabilities. We do run the risk that new flaws will be discovered and, since we are now one of many companies in the same "herd" running that vulnerable software, it is much more likely that our site will be compromised before we have an opportunity to patch it.

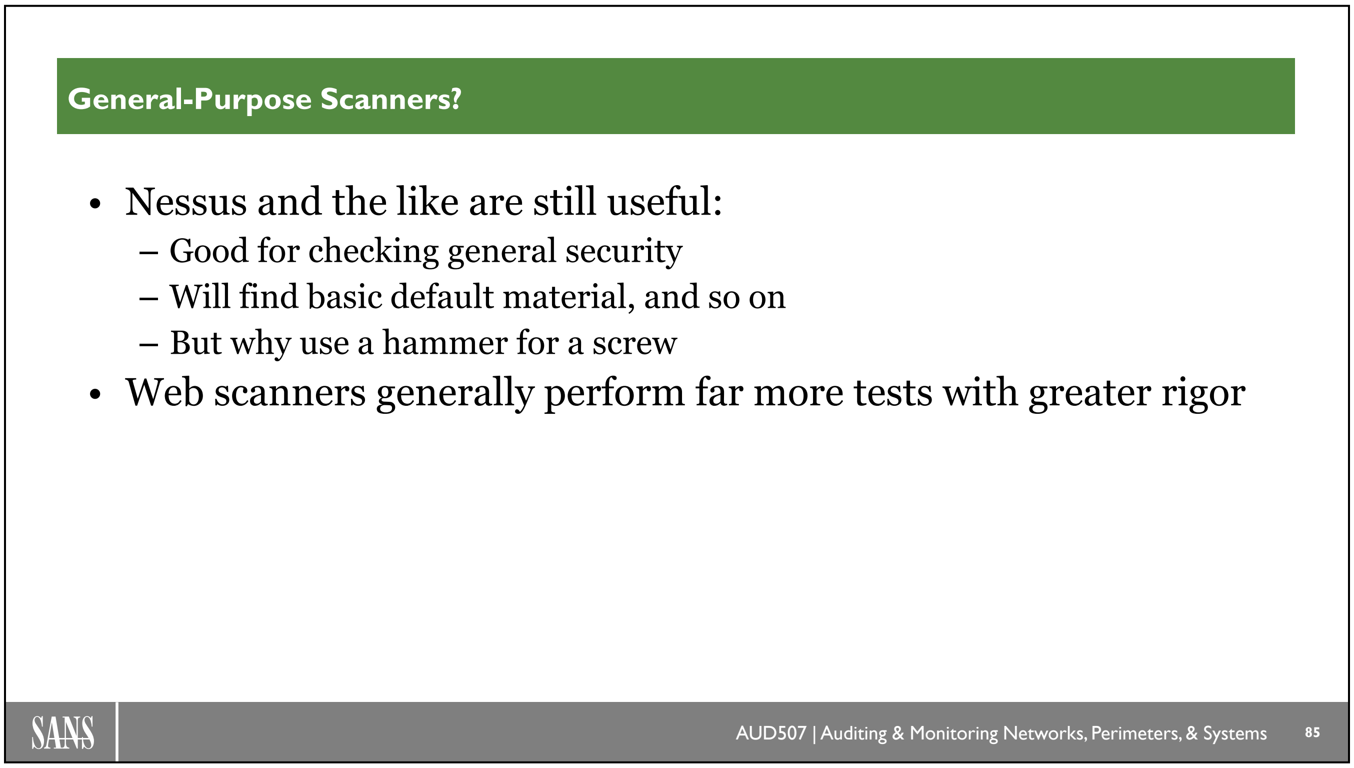
On the other hand, if we write our own application or customize an application, the vulnerability scanner likely won't find flaws in our application. This might mean that our application is more challenging to attack. Don't fall into a false sense of security, though. If you have a logic flaw or other authorization bypass in your application, it is extremely unlikely that an automated scanner will find it. Even though you can find it very easily, the automated scanner cannot tell that there's anything wrong because it is not smart enough to detect the logic error.

August 10, 2021

Therefore, we strongly recommend that you view these tools primarily as configuration validation tools. Anything more than configuration data is wonderful, but we likely need other tools to get those details.

© 2021 Risenhoover Consulting, Inc.
August 10, 2021

© SANS Institute 2021



General-Purpose Scanners?

- Nessus and the like are still useful:
  - Good for checking general security
  - Will find basic default material, and so on
  - But why use a hammer for a screw
- Web scanners generally perform far more tests with greater rigor

Some wonder if it is actually necessary to get a web-specific vulnerability scanning tool. Wouldn't a general-purpose vulnerability scanner be good enough? The answer is that the general-purpose scanners are still useful, especially for testing the configuration of the underlying operating system and network infrastructure, but the web-specific tools tend to do a much better job on web applications.

Web application vulnerability scanners usually have far more web-specific signatures and test web applications more rigorously than general-purpose scanners. This doesn't mean that you won't find one tool that does it all, but we recommend using the right tool for the job at hand.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

How Dynamic Scanners Work

- Typical strategy is to identify the server first:
  - Fingerprint the OS
  - Fingerprint the web service
  - Fingerprint add-ons (PHP, ColdFusion, and more)
- Test for server issues based on results:
  - Speeds things up
    - IIS 8 on Windows: MSXX-XXX attack
    - Apache on Linux: Chunked encoding attack

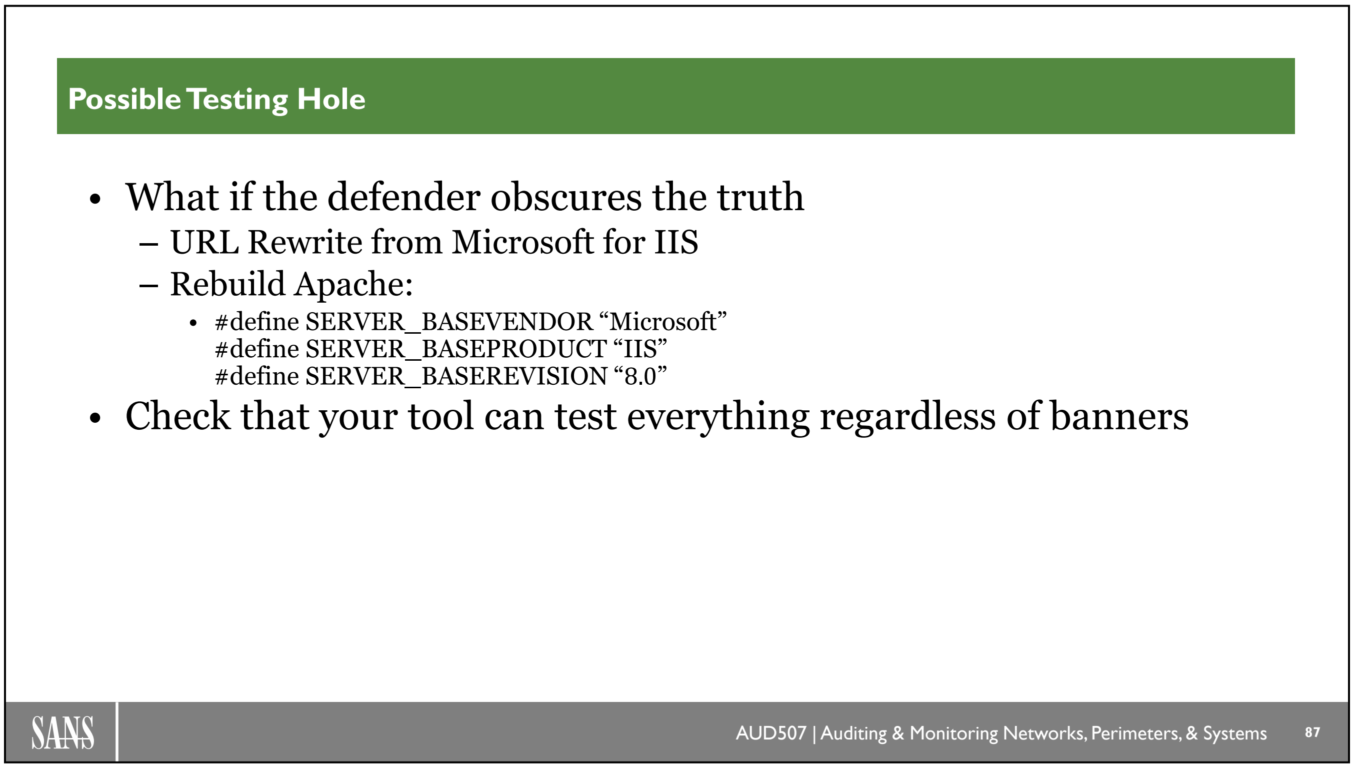
Vulnerability testing tools, though, tend to try to optimize their tests of the server in question. This is understandable because the tool will often have the capability of testing for hundreds or even thousands of vulnerabilities. Each one of these tests takes time. If the tool is used to target 10 or 20 systems, then this time grows exponentially. Imagine yourself running two testing tools side by side. They both come up with the same results, but one takes 3 minutes and the other takes 30 minutes. Which tool will you stick with?

As you can imagine this has had an impact on the market. Vendors are therefore focused not just on results, but on speed. However, the faster you go (the more shortcuts you take) the falser negatives you come up with. Knowing how these testing tools work helps us to understand how we will want to configure them to perform a thorough test.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Possible Testing Hole

- What if the defender obscures the truth
  - URL Rewrite from Microsoft for IIS
  - Rebuild Apache:
    - #define SERVER_BASEVENDOR "Microsoft"
      #define SERVER_BASEPRODUCT "IIS"
      #define SERVER_BASEREVISION "8.0"
- Check that your tool can test everything regardless of banners

We also take into account that any decent defender will at the least remove the banners from the services and some will even lie about the identity of the server. If you want to do this for your own servers or recommend it as a part of your security posture, there are tools and techniques available for all of the servers currently in use. For IIS, this is most easily done using Microsoft's URL Rewrite tool. If you use Apache, the most reliable way to replace the banner is to recompile the server, although some versions do allow you to modify the banner through the configuration file.

With these in mind, then, we also want to pick a testing tool that allows us to override the default fingerprinting results and test for everything regardless of the banners. This is also one of the reasons that manual testing is still so valuable for web application testing.

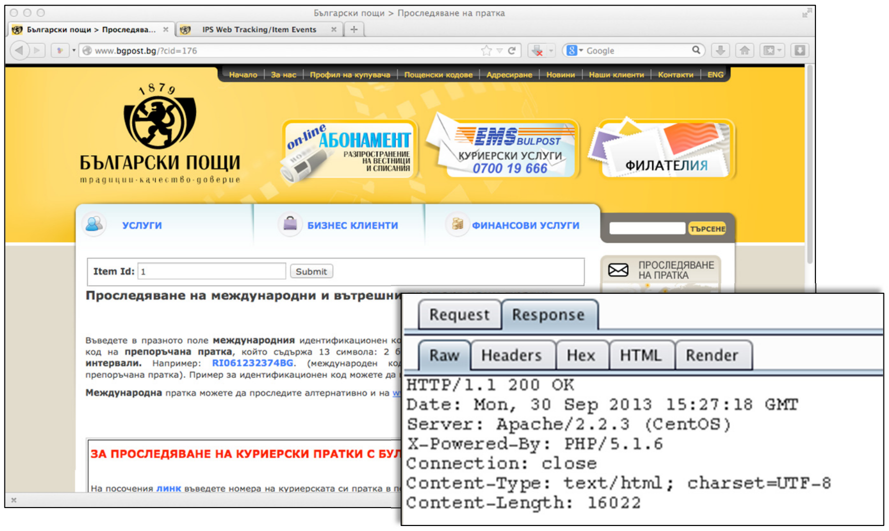
August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Is It Apache…



```
HTTP/1.1 200 OK
Date: Mon, 30 Sep 2013 15:27:18 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 16022
```

We may also discover that the information reported by the web server changes depending upon which page or other resource we request. Remember our discussion about cross domain requests and schizophrenic development? This is similar, but the change in platform may not be signaled by a switch to a new hostname or subdomain. Note in this slide that we are visiting the package tracking page for the Bulgarian postal service. The site appears to be running on an Apache server on a CentOS system. We can also see from the captured HTTP response that it is likely a PHP-based application.

If we were running an automated scanner, it would quite likely make the same identification that we have just made. But is it correct? Consider the next slide.

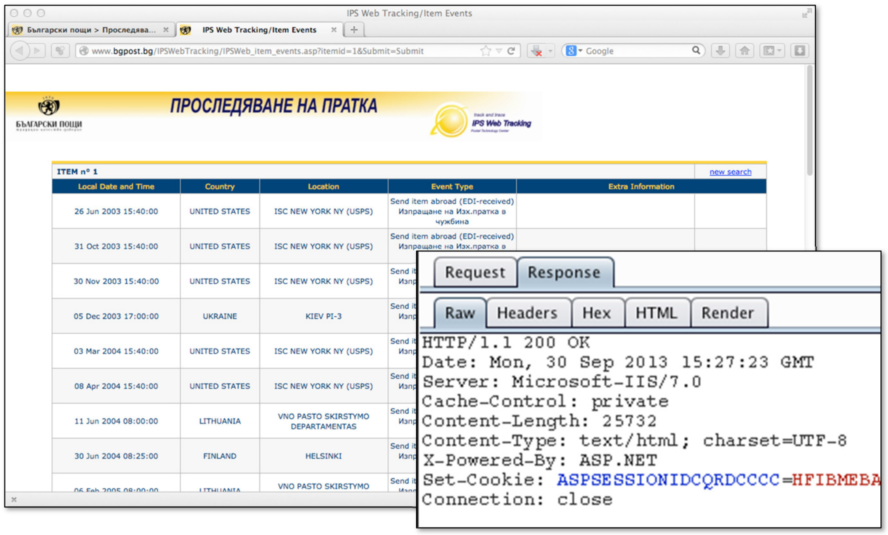
August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

…or IIS?



www.bgpost.bg/IPSWebTracking/IPSWeb_item_events.asp?itemid=1&Submit=Submit

ПРОСЛЕДЯВАНЕ НА ПРАТКА

ITEM nº 1 — new search

Local Date and Time	Country	Location	Event Type	Extra Information
26 Jun 2003 15:40:00	UNITED STATES	ISC NEW YORK NY (USPS)	Send item abroad (EDI-received) Изпращане на Изх.пратка в чужбина	
31 Oct 2003 15:40:00	UNITED STATES	ISC NEW YORK NY (USPS)	Send item abroad (EDI-received) Изпращане на Изх.пратка в	
30 Nov 2003 15:40:00	UNITED STATES	ISC NEW YORK NY (USPS)	Send it Изпр	
05 Dec 2003 17:00:00	UKRAINE	KIEV PI-3	Send it Изпр	
03 Mar 2004 15:40:00	UNITED STATES	ISC NEW YORK NY (USPS)	Send it Изпр	
08 Apr 2004 15:40:00	UNITED STATES	ISC NEW YORK NY (USPS)	Send it Изпр	
11 Jun 2004 08:00:00	LITHUANIA	VNO PASTO SKIRSTYMO DEPARTAMENTAS	Send it Изпр	
30 Jun 2004 08:25:00	FINLAND	HELSINKI	Send it Изпр	
06 Feb 2005 08:00:00	LITHUANIA	VNO PASTO SKIRSTYMO	Send it	

```
HTTP/1.1 200 OK
Date: Mon, 30 Sep 2013 15:27:23 GMT
Server: Microsoft-IIS/7.0
Cache-Control: private
Content-Length: 25732
Content-Type: text/html; charset=UTF-8
X-Powered-By: ASP.NET
Set-Cookie: ASPSESSIONIDCQRDCCCC=HFIBMEBA
Connection: close
```

At this point, we have asked it to search for tracking ID #1. You can see that the application is responding with a large number of results, but frankly, these results are completely immaterial to our point. Instead, take note of the URL and of the HTTP header that return!

You may notice that we are now at an ASP page. Even without this change in the URL, the embedded HTTP header indicates that we are speaking with a Microsoft IIS 7 server! The great part is that we are still speaking to the same host address. But what does this all mean for us?

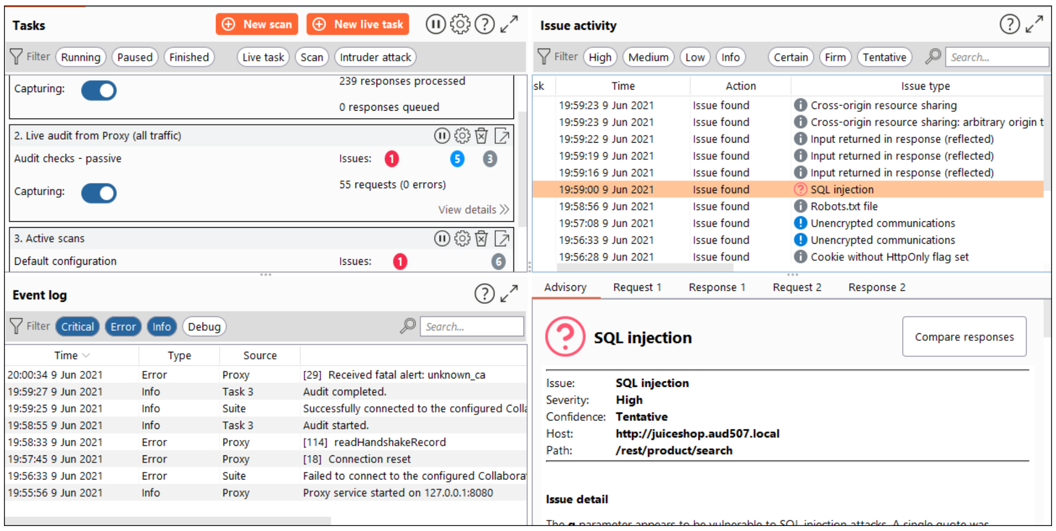
It is entirely possible that our scanner will have already determined that this is Apache on CentOS with PHP. Apparently, the server is somehow "proxying" some requests back to an IIS server. Will our scanner detect that the server technology has changed even though the IP address and hostname have not? If the scanner fails to note this change, then any report that it provides will be suspect because it may incorrectly fail to scan for certain types of issues or provide false positives for technologies that are not actually available over a specific URL!

August 10, 2021

August 10, 2021
Technet24

© SANS Institute 2021

Burp as a Scanner



```
Capturing:                         239 responses processed
                                   0 responses queued

2. Live audit from Proxy (all traffic)
Audit checks - passive             Issues: 1   5   3
Capturing:                         55 requests (0 errors)
                                                  View details

3. Active scans
Default configuration              Issues: 1   6
```

```
Time                Type    Source
20:00:34 9 Jun 2021 Error   Proxy   [29]  Received fatal alert: unknown_ca
19:59:27 9 Jun 2021 Info    Task 3  Audit completed.
19:59:25 9 Jun 2021 Info    Suite   Successfully connected to the configured Colla
19:58:55 9 Jun 2021 Info    Task 3  Audit started.
19:58:33 9 Jun 2021 Error   Proxy   [114]  readHandshakeRecord
19:57:45 9 Jun 2021 Error   Proxy   [18]  Connection reset
19:56:33 9 Jun 2021 Error   Suite   Failed to connect to the configured Collaborat
19:55:56 9 Jun 2021 Info    Proxy   Proxy service started on 127.0.0.1:8080
```

```
sk  Time                Action       Issue type
    19:59:23 9 Jun 2021 Issue found  Cross-origin resource sharing
    19:59:23 9 Jun 2021 Issue found  Cross-origin resource sharing: arbitrary origin t
    19:59:22 9 Jun 2021 Issue found  Input returned in response (reflected)
    19:59:19 9 Jun 2021 Issue found  Input returned in response (reflected)
    19:59:16 9 Jun 2021 Issue found  Input returned in response (reflected)
    19:59:00 9 Jun 2021 Issue found  SQL injection
    19:58:56 9 Jun 2021 Issue found  Robots.txt file
    19:57:08 9 Jun 2021 Issue found  Unencrypted communications
    19:56:33 9 Jun 2021 Issue found  Unencrypted communications
    19:56:28 9 Jun 2021 Issue found  Cookie without HttpOnly flag set
```

SQL injection

```
Issue:       SQL injection
Severity:    High
Confidence:  Tentative
Host:        http://juiceshop.aud507.local
Path:        /rest/product/search
```

Issue detail

The **q** parameter appears to be vulnerable to SQL injection attacks. A single quote was

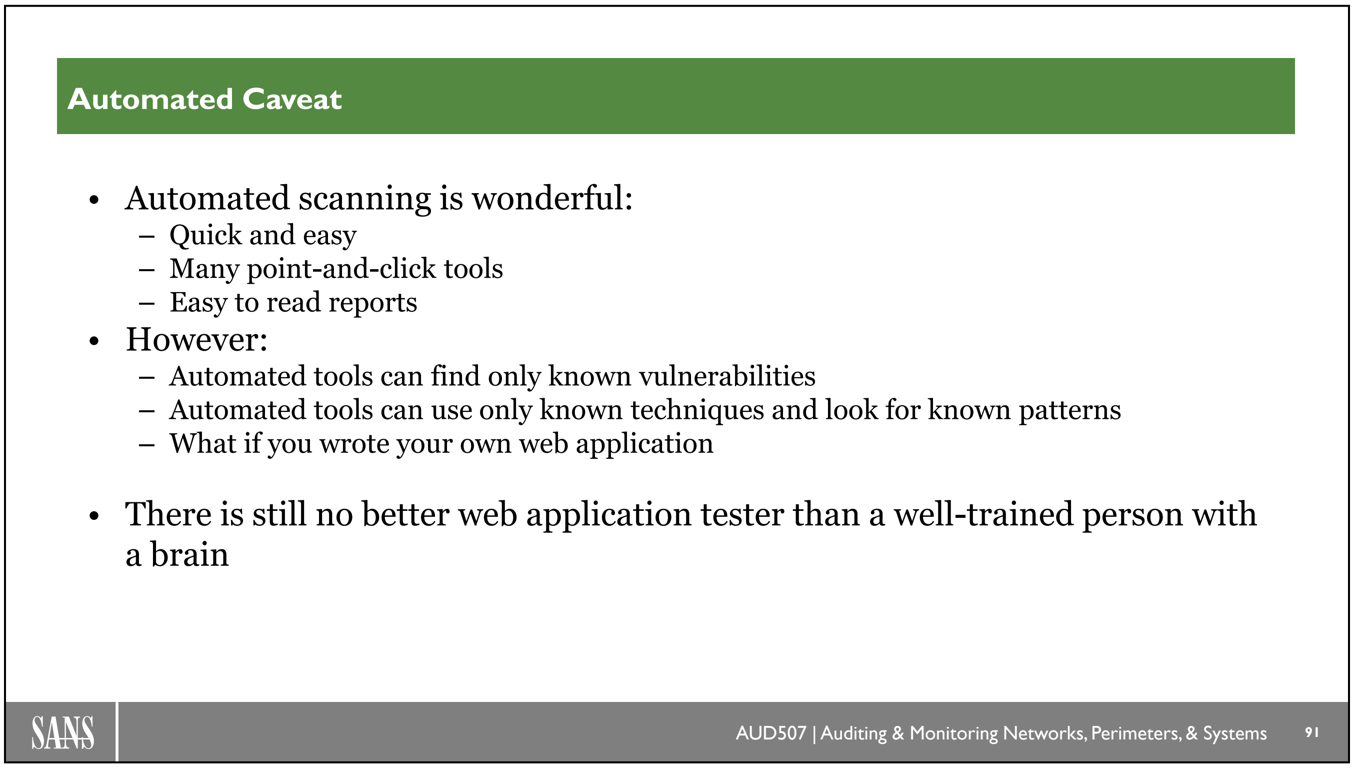
Among web application penetration testers and security professionals, Burp Suite is widely considered to be "the" tool for testing applications. In addition to having an extensible framework and a large number of built-in tools, the commercially licensed version has an effective vulnerability testing engine built into it.

The slide shows an example of the output from the Burp scanner. In this case, Burp was simply passively scanning content as it went by and adding notes about each thing that it proxied. Along the way, you can see that it actually has something to say about nearly every single website that the user has browsed to! Each of the items has detailed information to explain the issue. Each report includes both the content that was sent and precisely what was returned so that you can evaluate the data for yourself. To make the best use of this tool and what it reports, however, you must be familiar with what the actual impact would be and have at least a passing familiarity with how exploitation would be accomplished. We're going to seek to give you much of that knowledge throughout the remainder of the material in this section.

Burp is typically configured with a target scope. After doing so it is typical for a tester to tell Burp to automatically (and actively) scan everything that it comes across on the site. With this process up and running, it becomes a largely iterative process. The tester performs some manual testing. Burp watches and then actively scans this content. As things are found they are added to the report. Next, the tester finishes testing some page, element, or other aspect of the site, so he takes a peek at the report and selects something that Burp has found for validation and exploration.

August 10, 2021

© SANS Institute 2021



Automated Caveat

- Automated scanning is wonderful:
  - Quick and easy
  - Many point-and-click tools
  - Easy to read reports
- However:
  - Automated tools can find only known vulnerabilities
  - Automated tools can use only known techniques and look for known patterns
  - What if you wrote your own web application
- There is still no better web application tester than a well-trained person with a brain

There is a caveat with automated scanning that you ought to be aware of. We likely touched on this when we covered Nessus, but it would be good to make sure you keep this in mind.

Automated scanning is wonderfully quick, enabling you to easily configure the scanner to target however many hosts you need to test, producing a comprehensive report at the end. The problem, though, is that automated tools generally find only known vulnerabilities. This is great if you're running someone else's web application (such as a content management system or a blog), but what if your organization wrote its own web application?

This doesn't mean that automated tools aren't useful. They are great for finding the low-hanging fruit. Even so, there's no better web application tester today than a well-trained person with a brain. For some proof, you might try pointing Nikto at buggybank. It may find some issues, but it definitely won't find them all, and that's a badly written application!

August 10, 2021

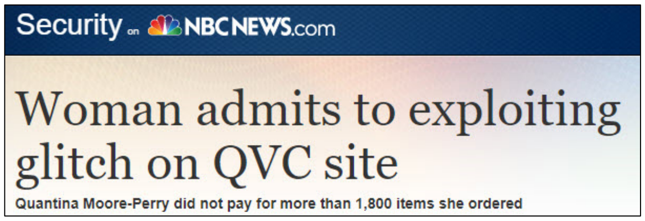
August 10, 2021

Technet24

© SANS Institute 2021

Business Logic Flaws

- Logic flaw allowed orders to be cancelled:
  - Credit card refunded
  - Item still shipped
- QVC alerted when eBay purchasers notified them
- *Would your automated scanner have found this flaw?*


Security on NBCNEWS.com

Woman admits to exploiting glitch on QVC site

Quantina Moore-Perry did not pay for more than 1,800 items she ordered

Moore-Perry pleaded guilty Thursday in federal court to wire fraud and was released pending sentencing. She agreed to forfeit the more than $412,000 she made from the scam.

This is an older story (it happened in 2007), but it's a perfect example of a critical logic flaw. A woman determined that if she ordered an item from QVC, then quickly canceled the order, she would receive a refund on her credit card. She would also still receive the merchandise she'd ordered! The web application and associated business processes had an exploitable logic flaw. When an order was canceled, it seems that no one told the warehouse not to ship the merchandise. She ordered around 1,800 items this way and then sold the items on eBay.

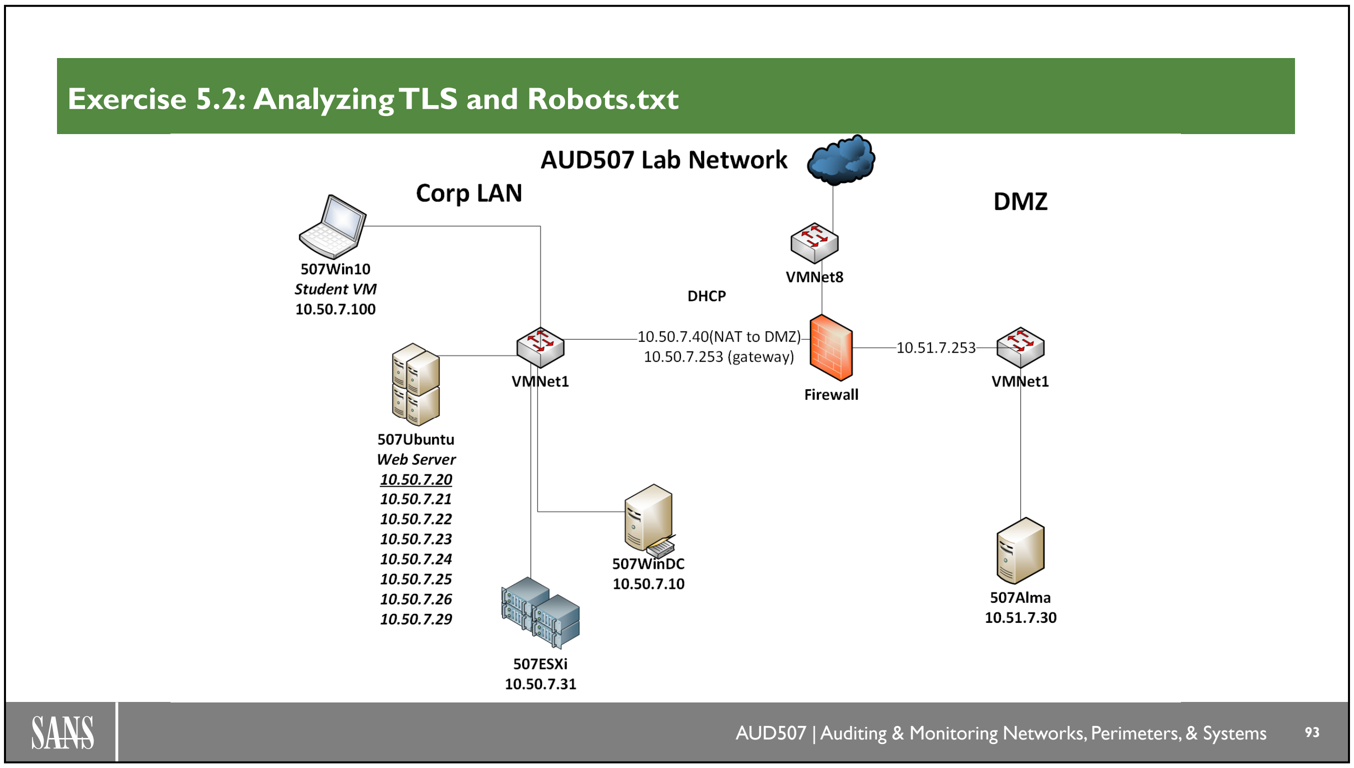
QVC found out about the scam when one of the eBay buyers let them know that there was a lady selling a LOT of QVC items on the site and that the items were coming still packaged in the original QVC packaging.

Web application scanners will not find this type of flaw in your applications. Only a well-trained person who understands the business process is likely to find logic errors such as this.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Exercise 5.2: Analyzing TLS and Robots.txt

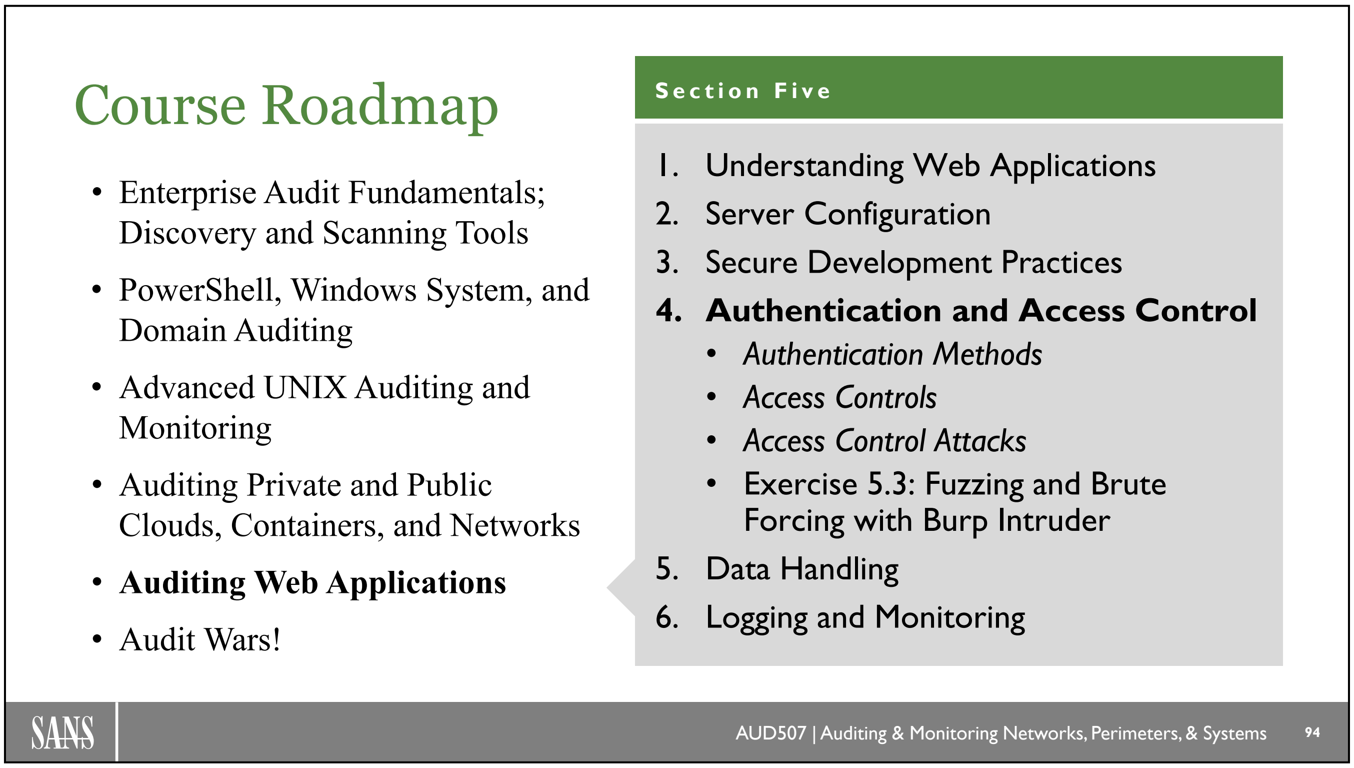
Please continue working in your workbook with Exercise 5.2: Analyzing TLS and Robots.txt.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Course Roadmap

- Enterprise Audit Fundamentals; Discovery and Scanning Tools
- PowerShell, Windows System, and Domain Auditing
- Advanced UNIX Auditing and Monitoring
- Auditing Private and Public Clouds, Containers, and Networks
- **Auditing Web Applications**
- Audit Wars!

Section Five

1. Understanding Web Applications
2. Server Configuration
3. Secure Development Practices
4. **Authentication and Access Control**
   - *Authentication Methods*
   - *Access Controls*
   - *Access Control Attacks*
   - Exercise 5.3: Fuzzing and Brute Forcing with Burp Intruder
5. Data Handling
6. Logging and Monitoring

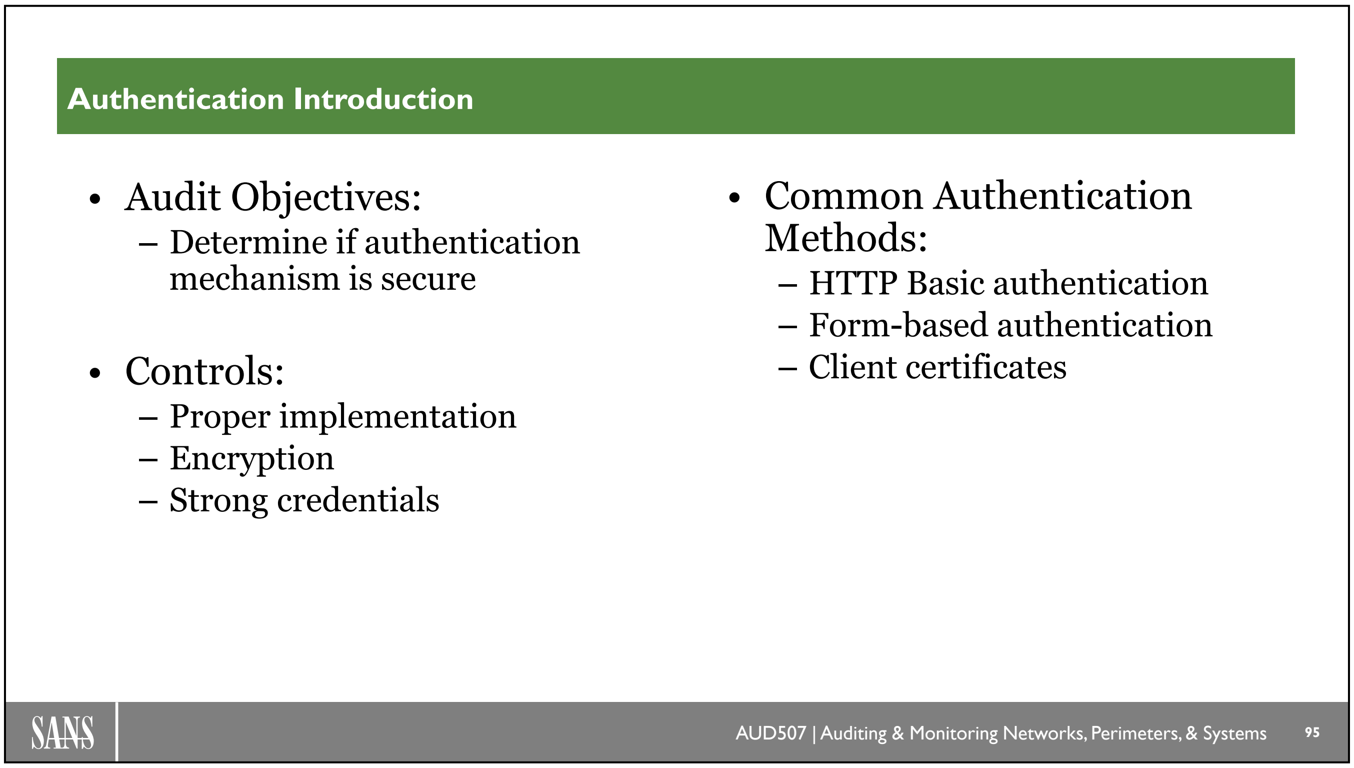
SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 94

This page intentionally left blank.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Authentication Introduction

- Audit Objectives:
  - Determine if authentication mechanism is secure
- Controls:
  - Proper implementation
  - Encryption
  - Strong credentials
- Common Authentication Methods:
  - HTTP Basic authentication
  - Form-based authentication
  - Client certificates

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 95

Many web-based applications authenticate the user (that is, validate the user's identity). If there are security weaknesses in the authentication, then the application is exposed to severe risk, such as unauthorized access, fraud, and theft of sensitive customer data. Let's examine several common forms of authentication and determine the relevant audit points.

There are three popular forms of authentication for web applications. These are the use of digital certificates, form-based authentication, and HTTP Basic authentication. We consider each of these, in turn, examining the pros and cons.

August 10, 2021

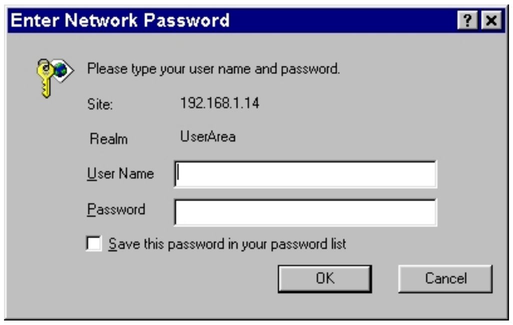
August 10, 2021

Technet24

© SANS Institute 2021

Methods: HTTP Basic Authentication

- Strengths:
  - Easy to implement
- Weaknesses:
  - Not easily cleared from browser (that is, logout function)
  - Not encrypted
  - Trivial to brute force
  - Acts like session ID:
    - Confuses long-term secret with short-term secret
- Best Practices:
  - Encrypt all traffic during and after authentication:
    - Credentials are sent with every request



Please type your user name and password.

Site:	192.168.1.14
Realm	UserArea
User Name	
Password	

☐ Save this password in your password list

HTTP has an authentication mechanism built into it. This is called Basic authentication. When a resource, such as a webpage, is protected by Basic authentication, the web server sends a special HTTP header. This header causes the browser to prompt the user for a username and password. The username and password are sent in an encoded form in the HTTP header of subsequent requests. This encoding is a base64 string to help comply with the HTTP specification, particularly removing spaces and other special characters. Encoding is not the same as encryption because it can easily be reversed without any special knowledge. One way to secure HTTP Basic authentication from eavesdropping is to encrypt the entire connection using TLS.

When Basic authentication is used, it is difficult to clear the credentials from the client. In fact, the easiest way to clear them is to convince the user to exit out of the browser completely! It is possible to implement a log-off function that clears the credentials by returning an HTTP 401 (Unauthorized) header to the client. To do this, however, you have to either automatically refresh to the 401 message or convince the user to click a sign-off button.

Before we move on from Basic authentication method, we should mention NTLM authentication. We won't spend any significant time on this topic because we are primarily focused on internet-facing applications. The security that NTLM authentication provides is considered "good enough" for internal applications as long as you accept whatever risks exist in your Windows Domain. The only significant risk inherent in NTLM authentication is that it can be vulnerable to all the session ID attacks that we discuss later unless the session tokens are protected (sent over a secure channel).

For now, here are some useful facts about NTLM authentication. This method is useful only for internal applications. The reason is that this method takes an internet standard (Hash-based authentication) and extends it. Another significant reason that limits our use of this for the internet is that Microsoft advises that a public-facing IIS server should never be joined to a domain for security reasons. If we aren't using a domain, why would we use domain accounts?

August 10, 2021

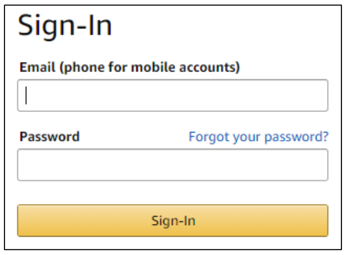
© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Authentication Methods: Form-Based

- Strengths:
  - Easy to implement
  - Reasonable balance between security and convenience for users
- Weaknesses:
  - Brute force potential
  - Misconfigurations could expose credentials
  - Same weaknesses as username/password
  - Not encrypted by default
- Best Practices:
  - Use HTTP POST to submit user credentials
  - Submission of user credentials is via encryption (e.g., HTTPS)
  - Password-type fields use TYPE=PASSWORD
  - Consider using tokens (e.g., SecureID)



The POST method is recommended for any form that submits sensitive data, especially authentication credentials. Because the GET method places all input (for example, the user's password) into the URL, this exposes the input in various ways (for example, the browser history file). More detailed examples of how the GET method exposes user input are given later.

The form should be submitted via HTTPS to prevent eavesdropping. The URL to which the form data is sent (the HTML parameter called ACTION) should be https (for encrypted), not simply http.

Form-based authentication should use the HTML tag of TYPE=PASSWORD for password-like form elements. This HTML tag causes user input to appear as asterisks (that is, *****) to prevent a malicious onlooker from seeing the sensitive data appear on the screen.

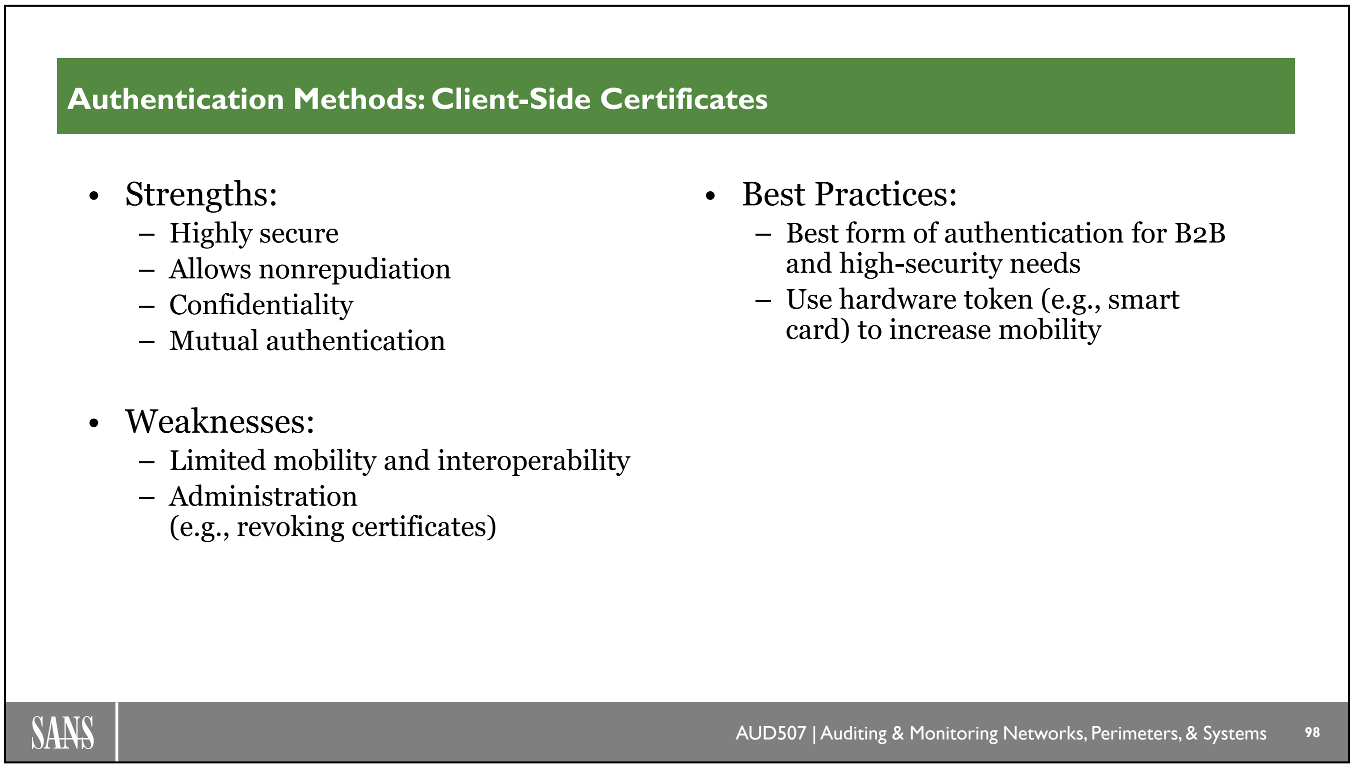
Finally, for the highest level of security based on forms, consider using tokens. Tokens are hardware devices or software applications that a user needs to successfully authenticate. Without physical possession of the token, it is (in theory) impossible for the user to log in.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Authentication Methods: Client-Side Certificates

- Strengths:
  - Highly secure
  - Allows nonrepudiation
  - Confidentiality
  - Mutual authentication
- Weaknesses:
  - Limited mobility and interoperability
  - Administration (e.g., revoking certificates)
- Best Practices:
  - Best form of authentication for B2B and high-security needs
  - Use hardware token (e.g., smart card) to increase mobility

Digital certificates are issued by trusted third parties known as Certification Authorities (CAs), using the industry-standard X.509 format. The CA digitally signs the certificate using its own private key, thereby vouching for the holder's identity and protecting the certificate against tampering.

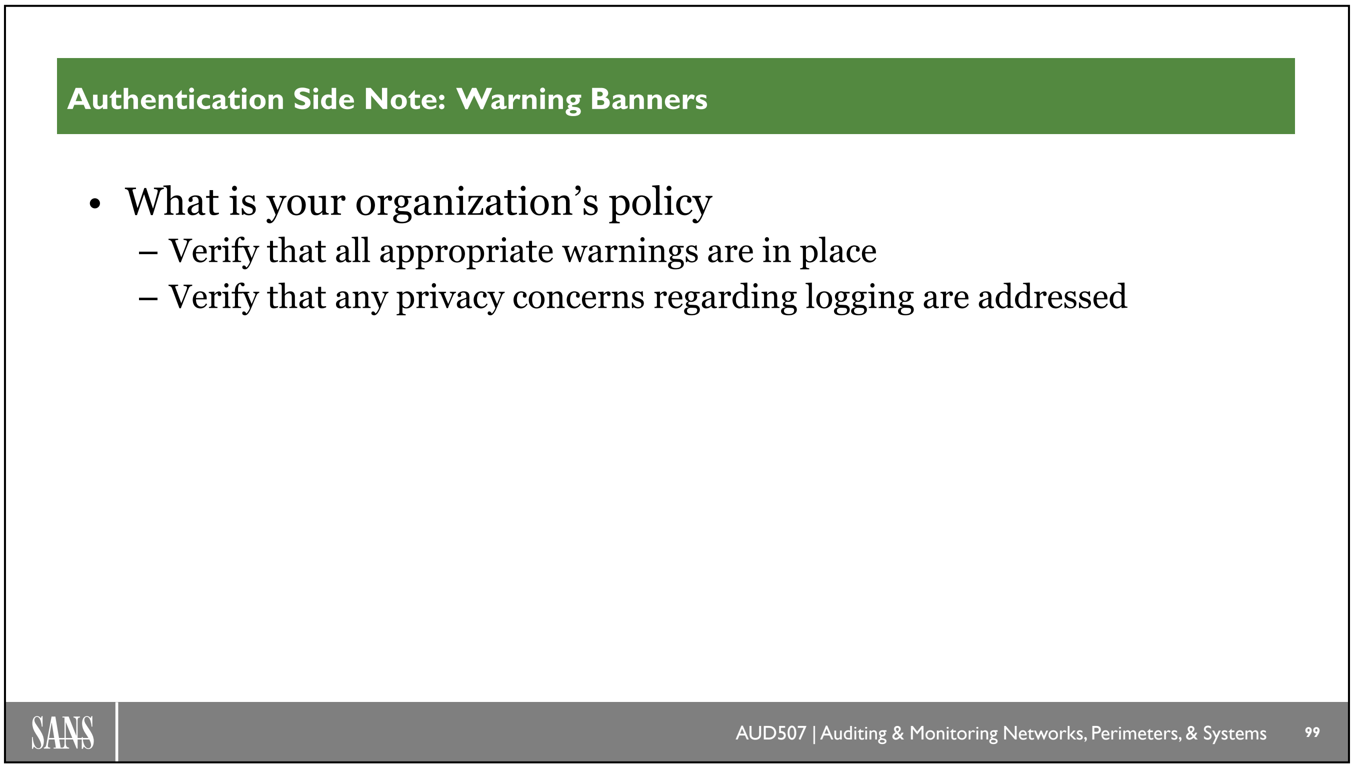
Digital certificates bind information about the certificate owner to the owner's public key, which is used to encrypt data. Typically, a web server that handles sensitive data encrypts, or scrambles, the traffic between the user and itself, especially when the data travels across untrusted networks, such as the internet. In that case, the web server presents the user with a digital certificate that serves two purposes:

- **Authentication:** The web server's identity is proven to the end user. This helps prevent users from being redirected to a malicious site surreptitiously.
- **Confidentiality:** The web server's public key is embedded within its digital certificate and facilitates the encryption of a session key, which is then used to encrypt traffic between the server and the user.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Authentication Side Note: Warning Banners

- What is your organization's policy
  - Verify that all appropriate warnings are in place
  - Verify that any privacy concerns regarding logging are addressed

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 99

The use of warning banners is generally encouraged for entry points into your web application, such as with login pages.

Your legal department should review and approve all warning banners. Verify that whatever your organization requires for warning banners has been appropriately displayed. It is also wise to research any privacy legislation issues that could impact any data collection that you perform and verify that you are adequately notifying users of what information you collect, why you collect it, and what recourse (if any) the consumer would have regarding that collection.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Username Harvesting (1)

- Threat: A malicious third party could collect valid usernames.
  - We reveal too much information:
    - Failed login
    - Account creation
    - Password reset options

A significant threat against web applications for authentication is brute force password-guessing attacks. To help to mitigate this threat, it is important to protect the usernames from discovery.

Frequently, to be helpful and to limit customer service calls, our application may provide far more information than is necessary. For example, when a user attempts to log on but enters the incorrect password, our application might offer a hint, a password reset option, or something similar. What if the user enters the wrong username? Does it present a different message? If so, an attacker needs to simply discover this behavior and then leverage it to begin harvesting usernames.

After a list of usernames has been generated, it is a simple matter to run a dictionary or brute force password-guessing attack against the application, potentially gaining unauthorized access!

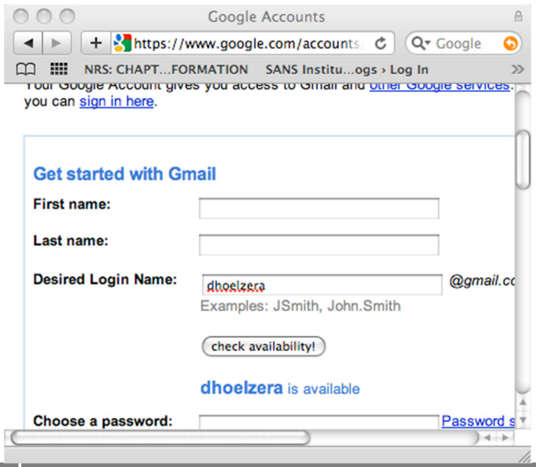
August 10, 2021

August 10, 2021

© SANS Institute 2021

Username Harvesting (2)

- Invalid Username



Google Accounts

https://www.google.com/accounts

you can sign in here.

Get started with Gmail

First name:

Last name:

Desired Login Name: dhoelzera @gmail.co

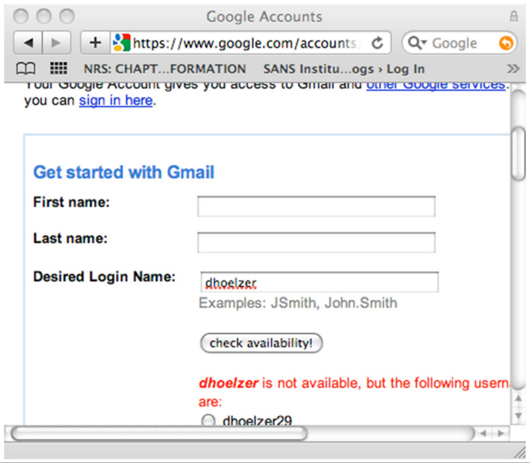
Examples: JSmith, John.Smith

check availability!

dhoelzera is available

Choose a password:

- Valid Username



Google Accounts

https://www.google.com/accounts

you can sign in here.

Get started with Gmail

First name:

Last name:

Desired Login Name: dhoelzer

Examples: JSmith, John.Smith

check availability!

dhoelzer is not available, but the following usern are:

dhoelzer29

For an example of this concept, look at the account creation process for Gmail. We are given the opportunity to select a logon name to use with Gmail. When we try **dhoelzera** and check availability, we find that we can use that name. If we try **dhoelzer**, we are informed that this name is *not* available!

This means that, with little effort, we can create a script that can harvest millions of usernames from Gmail in a short time. What can we do with them? Well, we could try some password attacks, but we could also sell the list to someone who sends unsolicited commercial email or other types of junk mail for a tidy profit!

Another potential impact of this is a denial-of-service attack. How so? If your system locks accounts out after a certain number of bad attempts, an attacker who harvests account names could simply choose to lock out all the accounts that have been discovered.

August 10, 2021

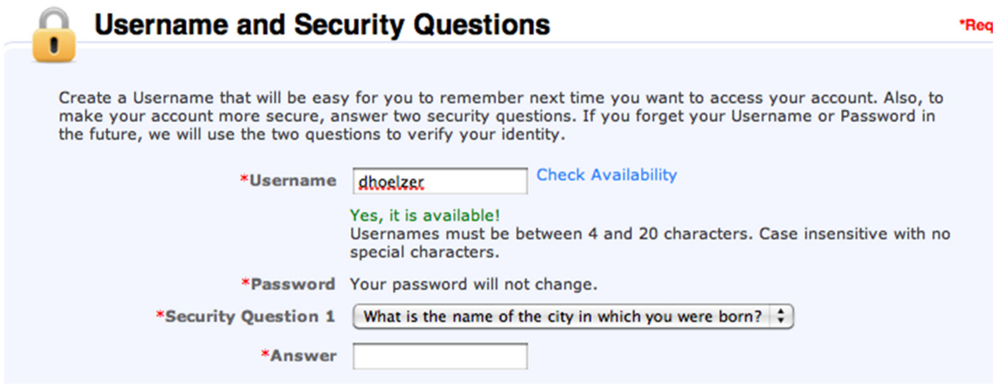
August 10, 2021

Technet24

© SANS Institute 2021

Username Harvesting (3)

- Southwest.com:



Username and Security Questions *Req

Create a Username that will be easy for you to remember next time you want to access your account. Also, to make your account more secure, answer two security questions. If you forget your Username or Password in the future, we will use the two questions to verify your identity.

*Username	dhoelzer Check Availability
	Yes, it is available! Usernames must be between 4 and 20 characters. Case insensitive with no special characters.
*Password	Your password will not change.
*Security Question 1	What is the name of the city in which you were born?
*Answer	

Just to be fair, here's another example of revealing usernames. Even if we don't show a lockout message or say, "That's not a valid username" versus "That's the wrong password" type messages, user account creation is a common place that we reveal too much information.

A common question that people ask in class is, "Okay. So how do we fix that? What if we want users to pick their own usernames?" Here's how we deal with this issue in our secure applications.

The user is given the opportunity to give input into what his username will be. However, regardless of what name the user selects, regardless of whether there is already a user with that name, the user is *never* permitted to have that name. Instead, he is then *assigned* a username based on what he has selected. For example, if the user selects **dhoelzer** our system will say, "That username is unavailable. We have assigned you username dhoelzer8321."

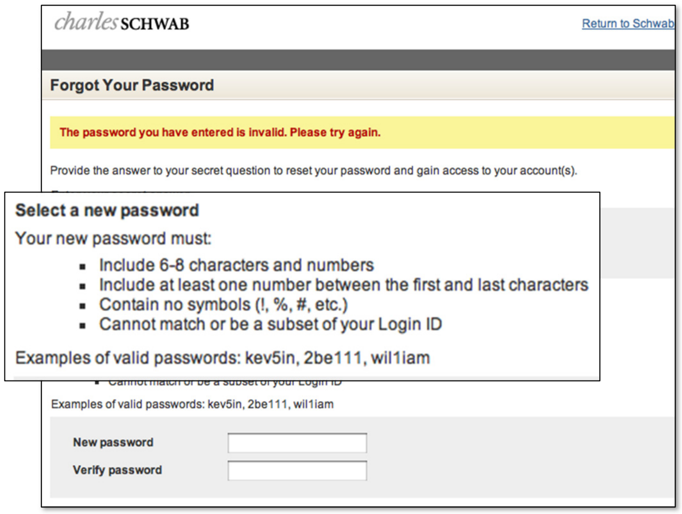
We are not saying that there are 8,320 other people with that username. The number tacked onto the end is a random value. Of course, we have verified that there is no other user who has already been assigned that particular username. In this way, the user has selected a piece of the username, but we have still created a mechanism that makes brute forcing the usernames *much* more difficult.

August 10, 2021

August 10, 2021

© SANS Institute 2021

On the Topic of Passwords



charles SCHWAB

Return to Schwab

Forgot Your Password

The password you have entered is invalid. Please try again.

Provide the answer to your secret question to reset your password and gain access to your account(s).

Select a new password

Your new password must:

- Include 6-8 characters and numbers
- Include at least one number between the first and last characters
- Contain no symbols (!, %, #, etc.)
- Cannot match or be a subset of your Login ID

Examples of valid passwords: kev5in, 2be111, wil1iam

Examples of valid passwords: kev5in, 2be111, wil1iam

New password	
Verify password	

As long as we're already talking about passwords, make sure that your applications allow your users to select strong passwords. Take note of this screenshot. This application was running on the Charles Schwab investment management site. The password that the user is selecting is used to protect investments, 401k funds, and more. Take note of the restrictions:

No longer than 8 characters long

No symbols of any kind

The only "strong" requirement is that a number must appear somewhere between the first and the last characters. Does this seem "strong" to you? Ensure that your applications *allow* your users to select strong passwords or phrases! Worse, the examples given by the website are quite likely to be used by frustrated users who do not completely understand the limitations!

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Audit Technique for Username Harvesting

- Testing:
  - Intentionally fail sign-on attempts, covering every possible scenario
  - Be careful not to lock any accounts; that comes later
  - Record all traffic (HTTP and HTML) and analyze for differences
  - If appropriate, test each language supported (e.g., English and German)
- Scenarios

User ID	PIN	Response
Valid	Invalid	A
Invalid	NA	B

- NOTE: Even the amount of time to return the error message may be an indicator

For the best results, compare all the HTML and HTTP (not just the visible error message) from each sign-on scenario. The error messages on the screen may be the same, but some HTML or HTTP (for example, cookie element) may be different, allowing usernames to be collected.

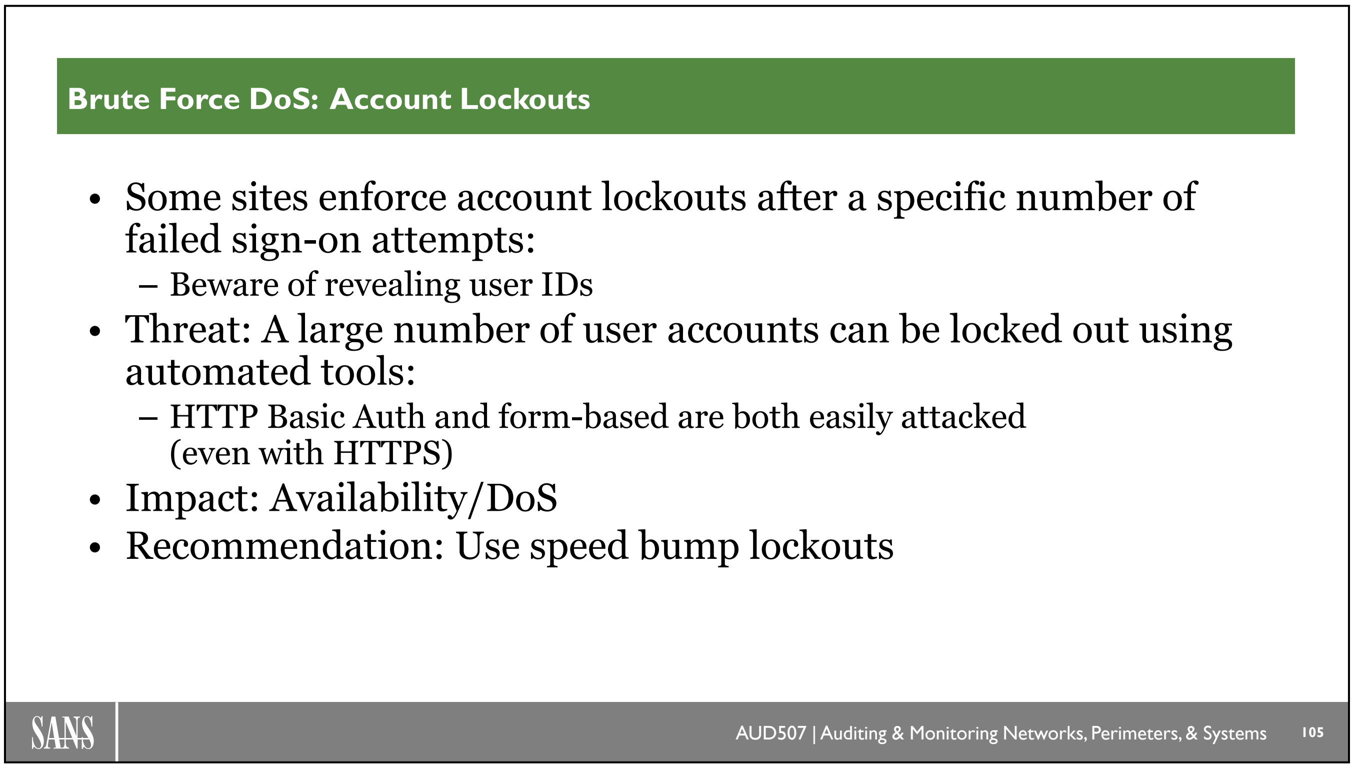
Another subtle indication could be the amount of time required to return the error message. Now, a lot of things can affect the web application's performance, so it is best to test this when you know the application is not loaded with other users, but it generally takes much longer to figure out that there is no entry in a SQL database than it does to find the correct entry when it exists.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Brute Force DoS: Account Lockouts

- Some sites enforce account lockouts after a specific number of failed sign-on attempts:
  - Beware of revealing user IDs
- Threat: A large number of user accounts can be locked out using automated tools:
  - HTTP Basic Auth and form-based are both easily attacked (even with HTTPS)
- Impact: Availability/DoS
- Recommendation: Use speed bump lockouts

Beware of locking accounts after a limited number of failed sign-on attempts. Freeware tools are readily available to assist a malicious third party in exploiting this to automatically lock out numerous accounts, creating a denial-of-service condition. How you protect against this is important because sometimes, the cure is worse than the disease.

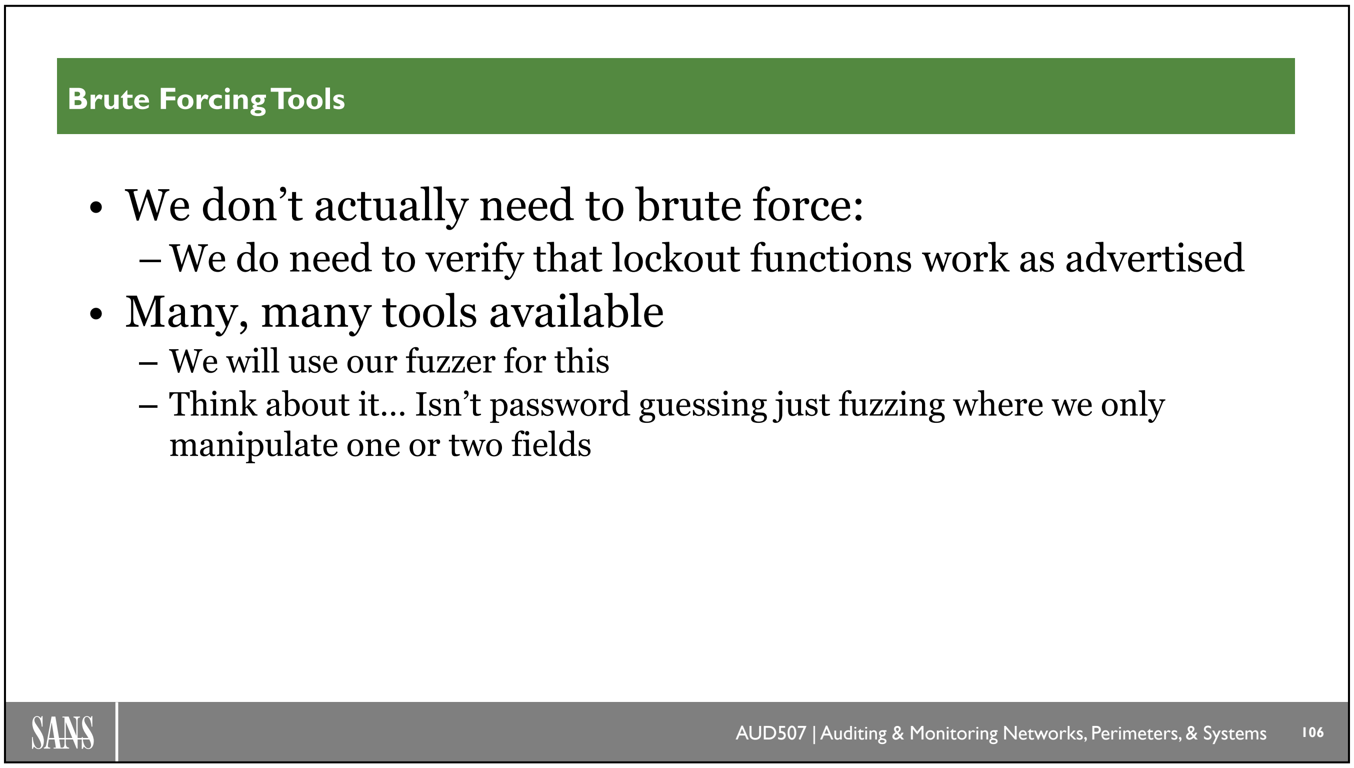
Implementing a lockout mechanism is important to prevent brute force attacks. However, how you implement a lockout mechanism is even more important. You should implement a "speed bump" mechanism whereby the web application inserts a delay, such as 30 seconds, between each login attempt. A variation on this theme is to increase the delay after consecutive failed login attempts. For example, after three incorrect attempts, the web application temporarily disables the account for 30 seconds. After the fourth login attempt is unsuccessful, the web application disables the account for another 45 seconds, and so on. This would continue until some maximum was reached, say 15 minutes. This makes brute forcing the password impractical, while still preventing a long-term DoS attack. During these delays, the attacker can continue to try passwords, but the application doesn't actually test any of them against the actual account!

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Brute Forcing Tools

- We don't actually need to brute force:
  - We do need to verify that lockout functions work as advertised
- Many, many tools available
  - We will use our fuzzer for this
  - Think about it… Isn't password guessing just fuzzing where we only manipulate one or two fields

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 106

There are a number of tools that can be used for brute force testing authentication and other forms. Before we discuss these in any depth, let's just mention that as an auditor we are typically not trying to actually break in; although this can happen. This is one of the areas where "Auditing" and "Penetration Testing" can cross paths.

It may be that we are trying to verify that password lockouts do occur. To determine this, we would likely begin by inquiring as to the lockout process. If validation is important for our objectives, then our next step might be to attempt to harvest out valid usernames. With these in hand, we might actually fire up a brute forcing tool like this to see whether the accounts do, in fact, lock out, how they unlock, what alerts are generated for the administrators, and so on. Clearly, the actual identities of the users are not particularly material to the outcomes.

There are some tools, such as Hydra and Brutus, that are designed specifically for brute forcing passwords in web applications. While these tools are fine, there's really no need for yet another tool. Think about what a fuzzer does. If you have completed that lab already, you know that a fuzzer can be used to repeatedly send queries, varying the parameters sent in the request. How does this apply to password guessing? In two ways.

If we have a list of usernames, we can use a fuzzer to fuzz the usernames using the list that we have collected and fuzzing the password with either a brute force approach or a dictionary. If we want to test a single username, we need simply set the username to be a static value and fuzz only the password.

August 10, 2021

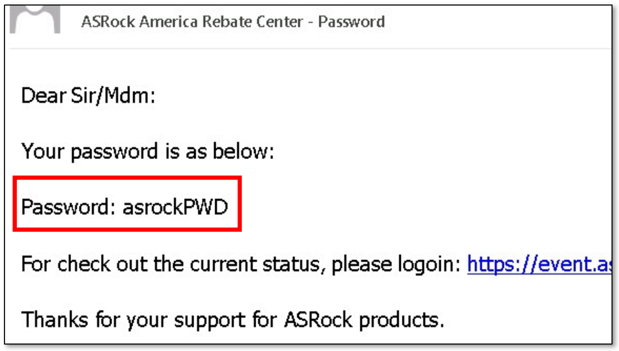
© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Authentication: Password Hashing

- Password storage controls:
  - Store only hashed passwords: no plaintext
  - Use salted hashes
  - Use iterative hashing algorithms (CPU time trade-off)



ASRock America Rebate Center - Password

Dear Sir/Mdm:

Your password is as below:

Password: asrockPWD

For check out the current status, please logoin: https://event.as

Thanks for your support for ASRock products.

User passwords stored in the applications database should always be protected with some sort of cryptography. Some simple rules to follow are:

- Never store plaintext passwords. Use a cryptographic (one-way) hashing algorithm to protect passwords in storage.
- Use "salts" – random data prepended to the user's password before encryption – to protect against pre-computed password attacks such as rainbow tables.
- Consider the use of iterative hashing functions, which perform multiple (sometimes as many as thousands) of rounds of hashing on the password. This approach makes password cracking more difficult, but it also requires more CPU time on the server. My recommendation is that iterative hashing should never take more than .25 seconds.

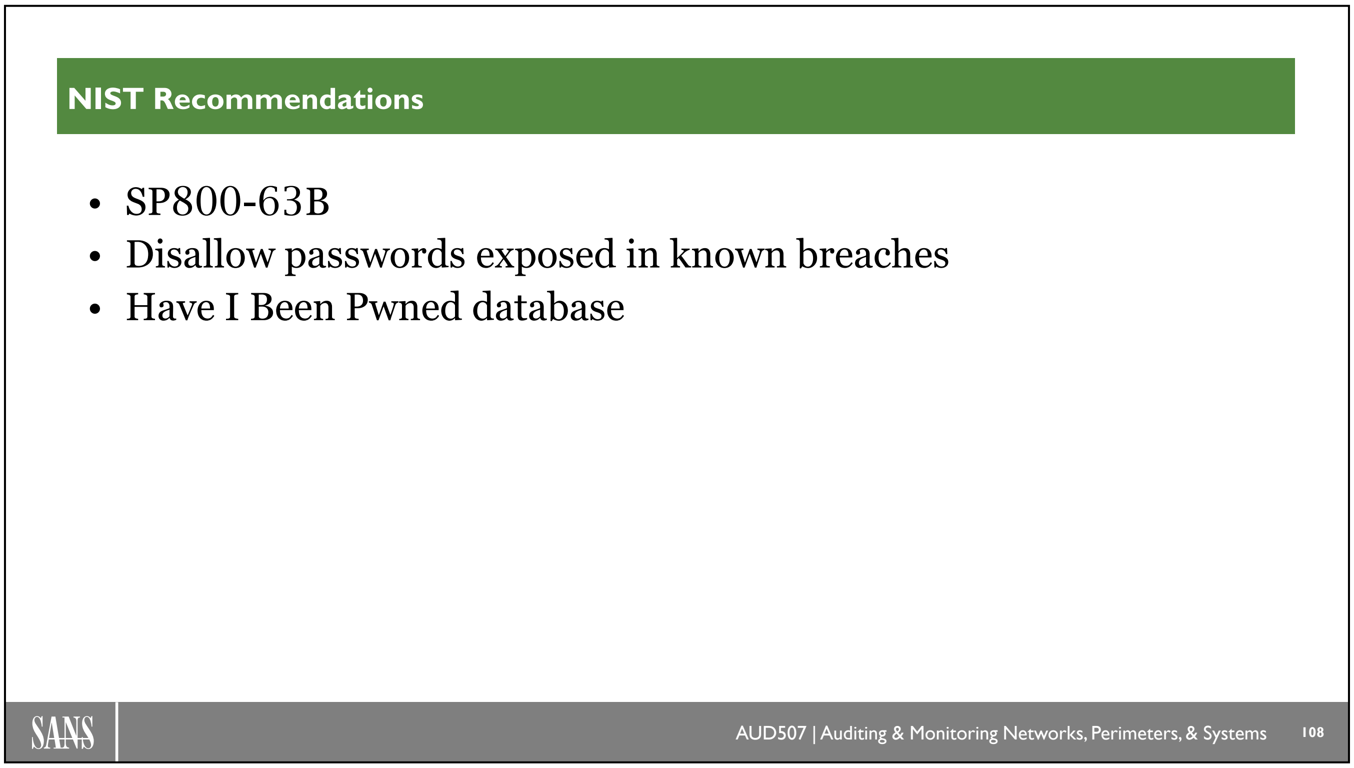
Note the plaintext password in the email on the slide. It is apparent that the web application which generated this email does not encrypt users' passwords.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



NIST Recommendations

- SP800-63B
- Disallow passwords exposed in known breaches
- Have I Been Pwned database

NIST recommends in its Digital Identity Guidelines document, NIST SP800-63B, that web applications disallow the use of passwords which are known to have been exposed in previous data breaches. Following this recommendation would require a site to keep a database (NIST calls it a corpus) of breached passwords.

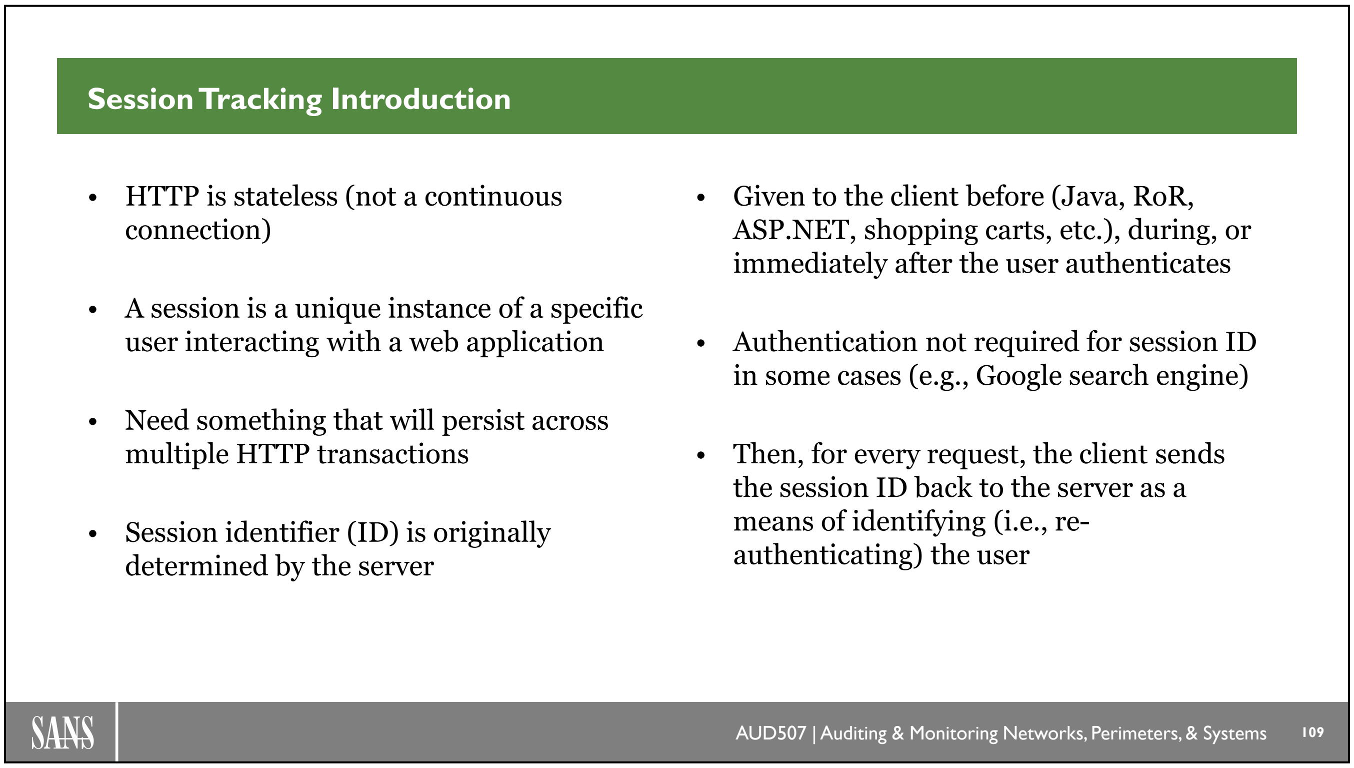
Fortunately, Troy Hunt at haveibeenpwned.com has developed just such a database! The database is available for query through a web-based API and uses hashing and partial-hash searches to preserve the confidentiality of passwords being tested.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Session Tracking Introduction

- HTTP is stateless (not a continuous connection)
- A session is a unique instance of a specific user interacting with a web application
- Need something that will persist across multiple HTTP transactions
- Session identifier (ID) is originally determined by the server
- Given to the client before (Java, RoR, ASP.NET, shopping carts, etc.), during, or immediately after the user authenticates
- Authentication not required for session ID in some cases (e.g., Google search engine)
- Then, for every request, the client sends the session ID back to the server as a means of identifying (i.e., re-authenticating) the user

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 109

This section focuses on the method by which the server/application tracks the user's session and the security implications.

A "session" is a unique instance of a user throughout the course of his interaction with the application. Authentication may not be needed by the website, yet a session may exist – for example, on a site that will allow you to "shop" and then worry about who you are (or even allow you to check out as a guest) after you have made your shopping choices. If authentication is required, then protecting session tracking becomes even more critical.

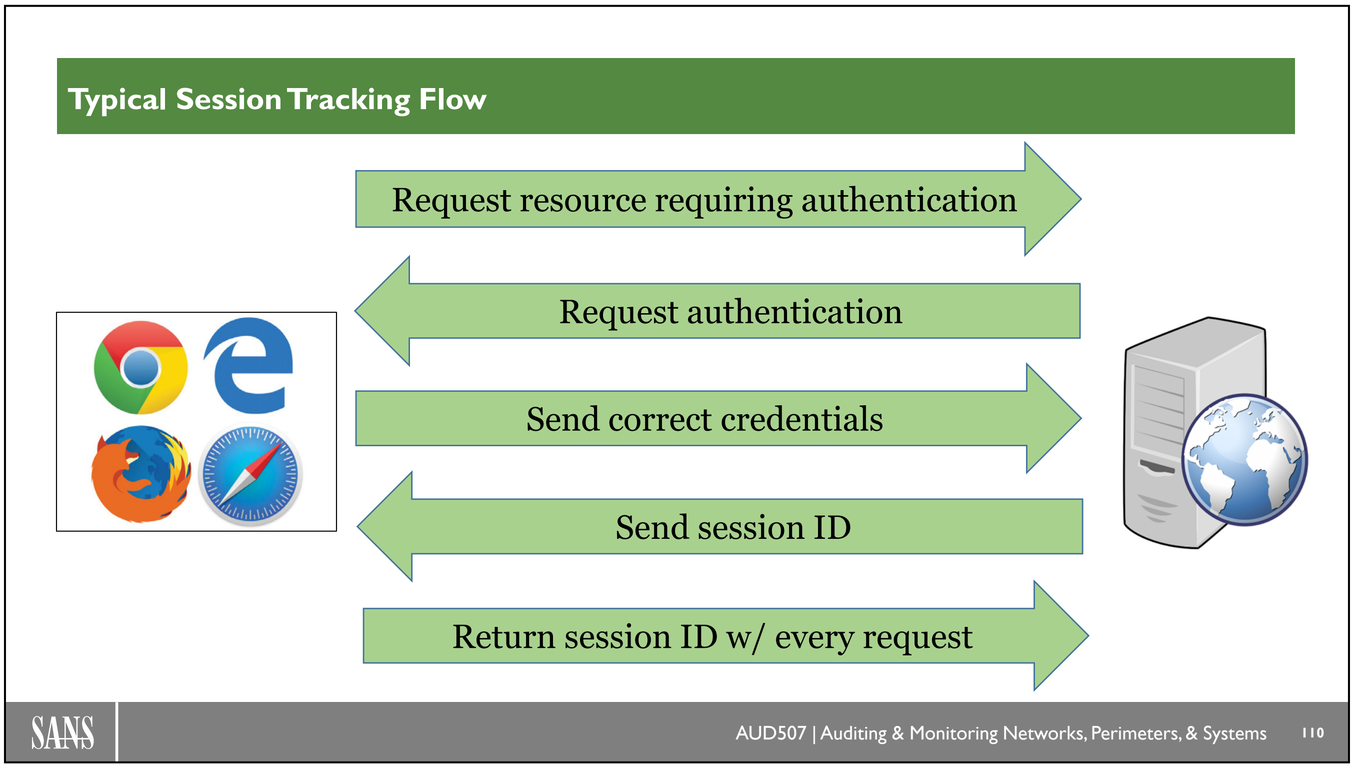
The key concept to understand for session tracking is that a unique identifier (which we call the session ID) is used to identify the user for every request made by the web browser. If the site requires authentication, then this session ID acts as a form of continuous re-authentication. Anyone with your session ID can impersonate you when speaking to the web application. This is poorly understood by many developers, but is a key aspect of web application programming, since the HTTP protocol itself is stateless.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Typical Session Tracking Flow

Request resource requiring authentication
Request authentication
Send correct credentials
Send session ID
Return session ID w/ every request

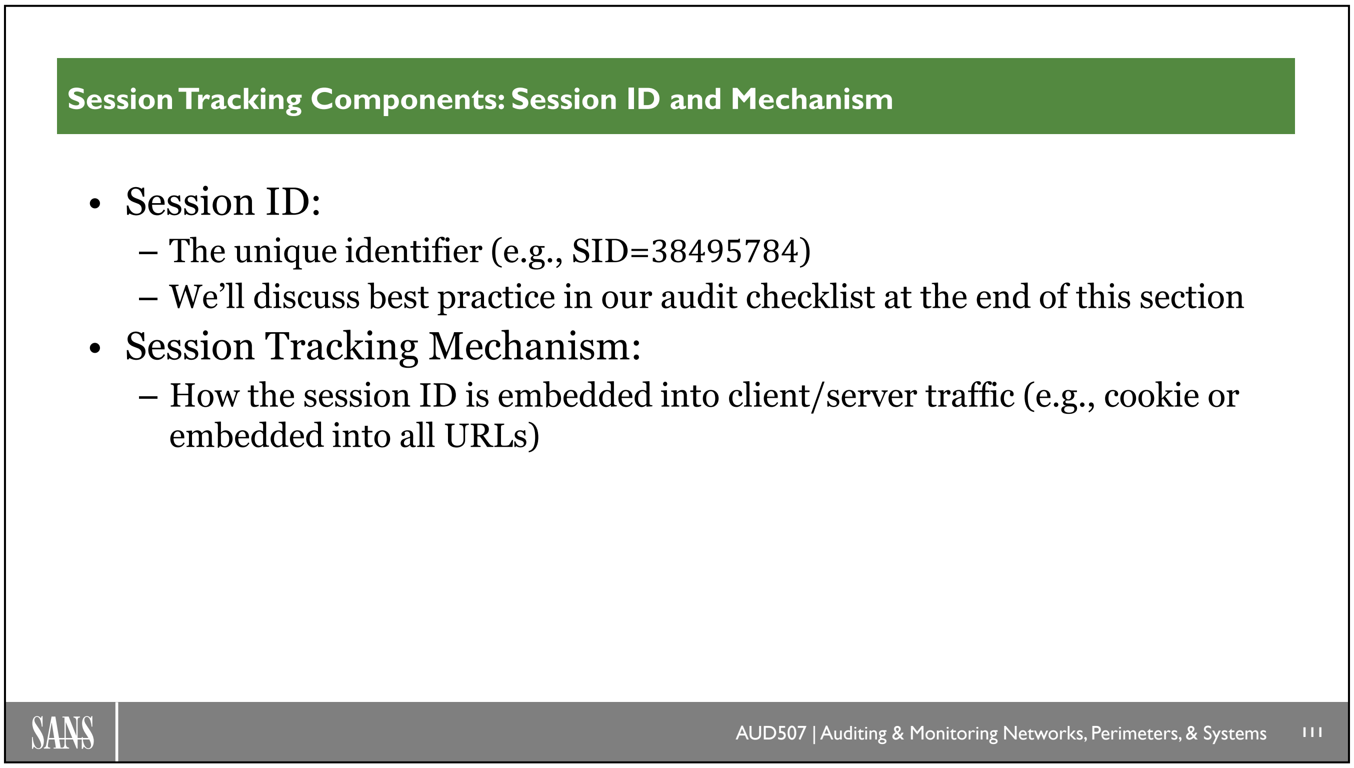
This is an overview of the session-tracking flow of an average web application.

- At some point during the session, the client will request a resource which requires authentication.
- The server will request credentials from the client. This could be by redirecting to a login form or by using the WWW-Authenticate header to request basic authentication.
- Once the client has supplied correct credentials, the server will issue a session ID. The ID could be sent in a cookie, written into the URL, added to a hidden form field, or in the case of HTTP basic authentication will consist of the encoded username and password which will be sent with each request.
- The client will send the session ID back to the server with subsequent requests to prove that the user is authenticated and allowed to perform the actions they request.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Session Tracking Components: Session ID and Mechanism

- Session ID:
  - The unique identifier (e.g., SID=38495784)
  - We'll discuss best practice in our audit checklist at the end of this section
- Session Tracking Mechanism:
  - How the session ID is embedded into client/server traffic (e.g., cookie or embedded into all URLs)

Two primary areas must be considered when examining a web application that maintains state. First, there is the session ID. We must examine how it is created and managed over time. The creation and management of the session ID is a significant area for analysis.

The second is the session tracking mechanism in use. This is the mechanism used to transfer the session ID between the client and the server. Frequently, you will find that in this context the session ID is encapsulated in some way to make up a composite session token. Using the incorrect or an insecure session tracking mechanism is the other significant area that requires our attention.

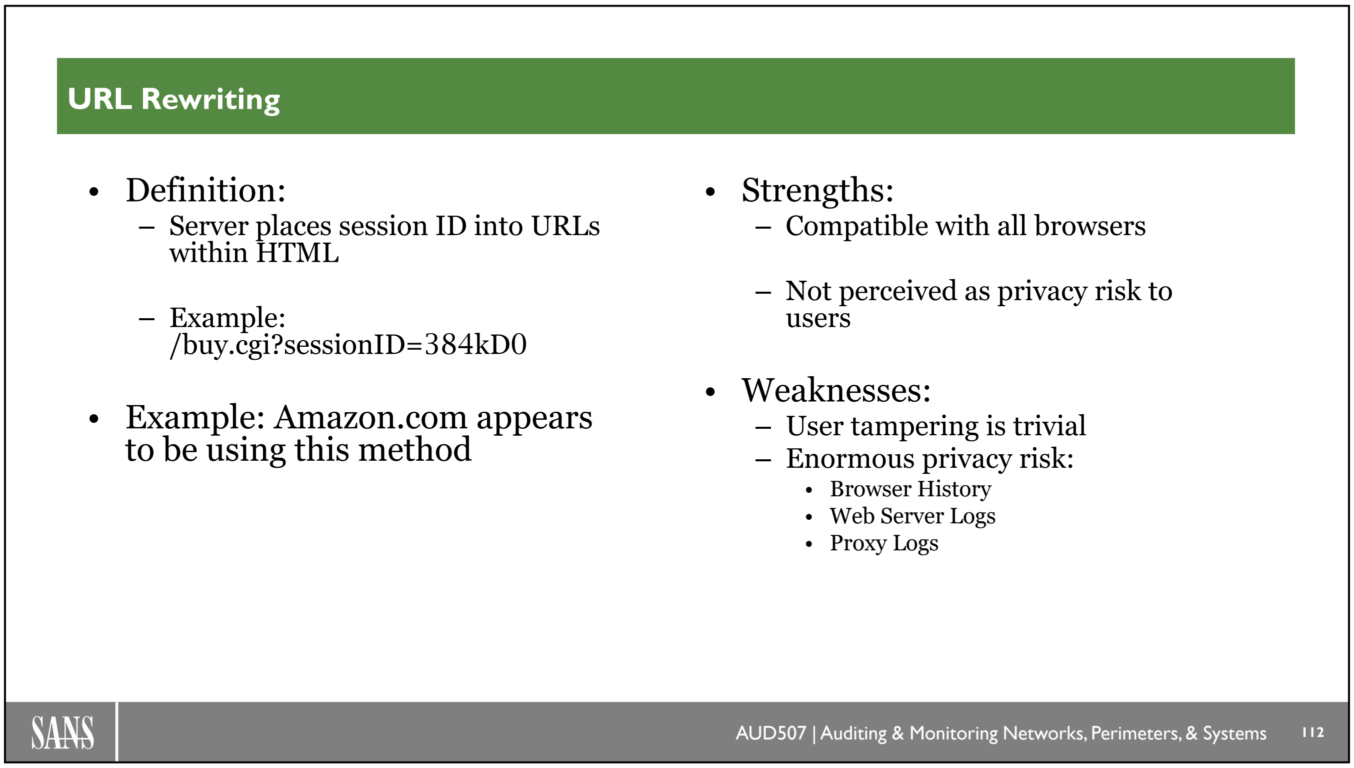
We'll look at the main techniques used for session management and discuss how to generate good session IDs as well as how to identify bad ones.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



URL Rewriting

- Definition:
  - Server places session ID into URLs within HTML
  - Example: /buy.cgi?sessionID=384kD0
- Example: Amazon.com appears to be using this method
- Strengths:
  - Compatible with all browsers
  - Not perceived as privacy risk to users
- Weaknesses:
  - User tampering is trivial
  - Enormous privacy risk:
    - Browser History
    - Web Server Logs
    - Proxy Logs

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 112

A key concept for each session tracking method will be that the session ID is to be considered sensitive data, the short-term secret, and therefore must be protected from accidental exposure.

The browser's history file is stored in cleartext and may be susceptible to remote theft due to browser weaknesses or physical access. Therefore, URLs are not best suited for transmitting sensitive data, such as session IDs. URLs may also be revealed in the HTTP Referer field. Finally, session IDs may be accidentally placed at the end of non-encrypted URLs. Users clicking such a link would unknowingly transmit their session ID across the internet unencrypted.

On the flip side, the advantages of using URL rewriting include compatibility with all browsers, and the fact that users do not generally perceive this method as a privacy issue.

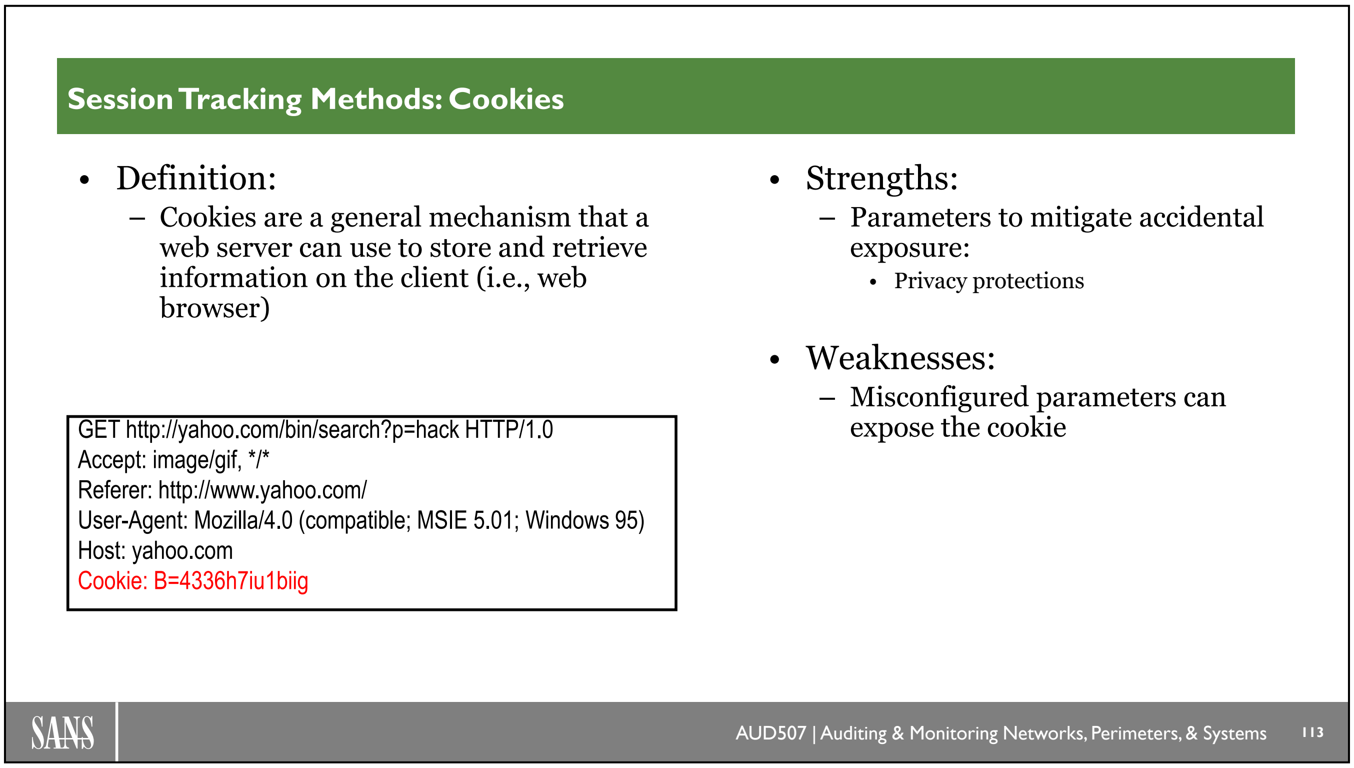
Using URL rewriting may be acceptable but consider carefully how the risks are mitigated in the application to account for exposure points.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Session Tracking Methods: Cookies

- Definition:
  - Cookies are a general mechanism that a web server can use to store and retrieve information on the client (i.e., web browser)

```
GET http://yahoo.com/bin/search?p=hack HTTP/1.0
Accept: image/gif, */*
Referer: http://www.yahoo.com/
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows 95)
Host: yahoo.com
Cookie: B=4336h7iu1biig
```

- Strengths:
  - Parameters to mitigate accidental exposure:
    - Privacy protections
- Weaknesses:
  - Misconfigured parameters can expose the cookie

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 113

We already covered cookies, but let's take a closer look. Remember, cookies are nothing more than small pieces of text sent between web servers and clients. They provide a convenient place for the web server to store a session ID.

Cookies have various fields that allow the server to dictate how the client should handle the cookie. These, when used properly, can help reduce the changes of the session ID from being compromised. For example, marking the cookie as "Secure" requires that HTTPS is in use for the cookie to be sent. Setting an expiration date provides for the capability to limit the lifetime or even make the cookie nonpersistent.

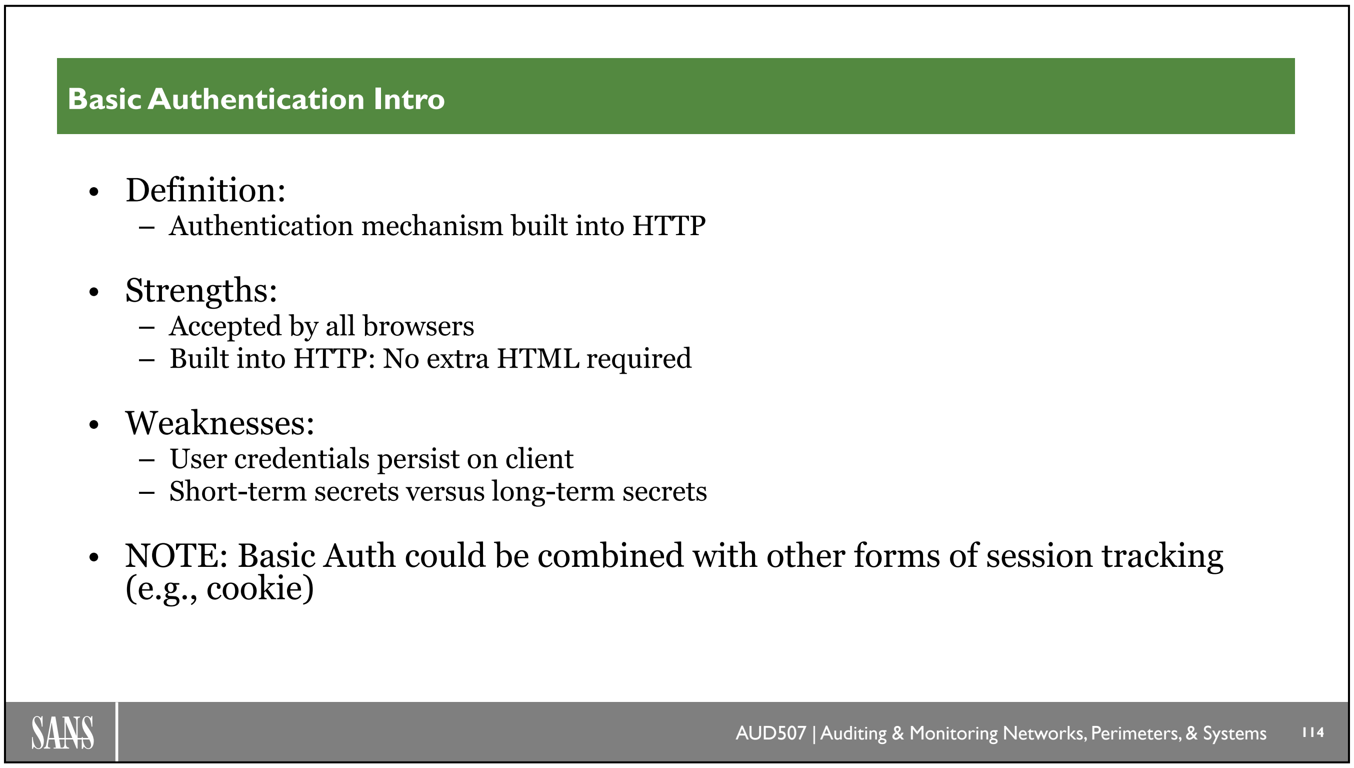
Here's an interesting fact: The reason that cookies were added to the HTTP standard was to allow application developers to maintain state. This means that if you want to maintain state, cookies should be the right way to do it! Why are people using other methods like URL rewriting? The reason is that users used to perceive cookies to be a serious privacy risk. Some users even completely disabled cookies. Older web applications may still use other mechanisms because of these historical concerns.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Basic Authentication Intro

- Definition:
  - Authentication mechanism built into HTTP
- Strengths:
  - Accepted by all browsers
  - Built into HTTP: No extra HTML required
- Weaknesses:
  - User credentials persist on client
  - Short-term secrets versus long-term secrets
- NOTE: Basic Auth could be combined with other forms of session tracking (e.g., cookie)

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 114

With Basic authentication, the user's credentials persist in the browser's memory. There are ways for an attacker (with physical access to the client) to extract these credentials. This technique also means that we are using the long-term secret as the short-term secret.

On the positive side, this form of authentication is supported by every single browser ever made. An enormous risk, though, is that no server that we know of has built-in support for detecting and locking accounts using Basic authentication, should brute force login attempts occur.

Also remember that the credentials will be sent in every single request to that web server until you either close the browser completely or send a 401 message back to the browser.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.
August 10, 2021

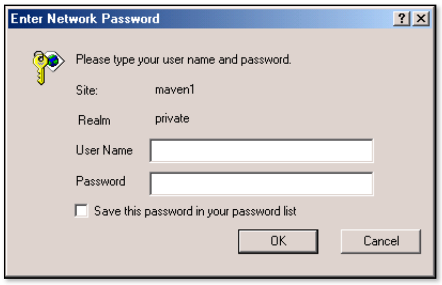
© SANS Institute 2021

Traffic Flow for Basic Auth (1)

1. Browser requests a protected resource
2. Server sends response that tells browser to use Basic Auth
3. Browser sees `WWW-authenticate` header and generates a Basic Auth prompt

```
HTTP/1.1 401 Unauthorized
Server: Xitami
Date: Tue, 23 Apr 2001 16:13:22 GMT
WWW-authenticate: Basic realm="private"
Content-length: 107
Content-type: text/html

<HTML><TITLE>Error</TITLE><BODY><H1>
Your username and/or password are invalid.
</H1></BODY></HTML>
```


Enter Network Password
?
Please type your user name and password.
Site: maven1
Realm private
User Name
Password
Save this password in your password list
OK
Cancel

When you use your browser to go to a website that requires Basic authentication, the server will send back an HTTP 401 message, which means that you have not properly authenticated. Included within that message will be a header, "www-authenticate." This header, coupled with the "Basic" identifier, causes your browser to open the username/password dialog box seen in the slide. What happens when you enter your username and password into these fields?

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Traffic Flow for Basic Auth (2)

4. After user enters his username and password, his browser sends them base64 encoded with a colon (:) between them:

```
GET /private/private.htm HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
excel, application/msword, application/vnd.ms-powerpoint, */*
Referer: http://enclave.com/
Accept-Language: en-us
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0; T312461)
Host: score.auditcasts.com
Authorization: Basic SGV5OnF1aXQgcGVla2luZyE=
```

When you enter your credentials and submit them, the web browser encodes the username and password and embeds the result into another HTTP header for the request sent to the server. You can see this at the bottom of the slide where the "Authorization" header appears.

We know that it looks like that password is encrypted, but it isn't. This is simply base64 encoded, just like email attachments are encoded. Why are the username and password encoded? To avoid violating HTTP special character rules. For example, your password might contain a colon. Attempting to embed that directly into the client request would interfere with HTTP because HTTP uses the colon character to separate HTTP header names from their values. For example, Server**:** IIS.

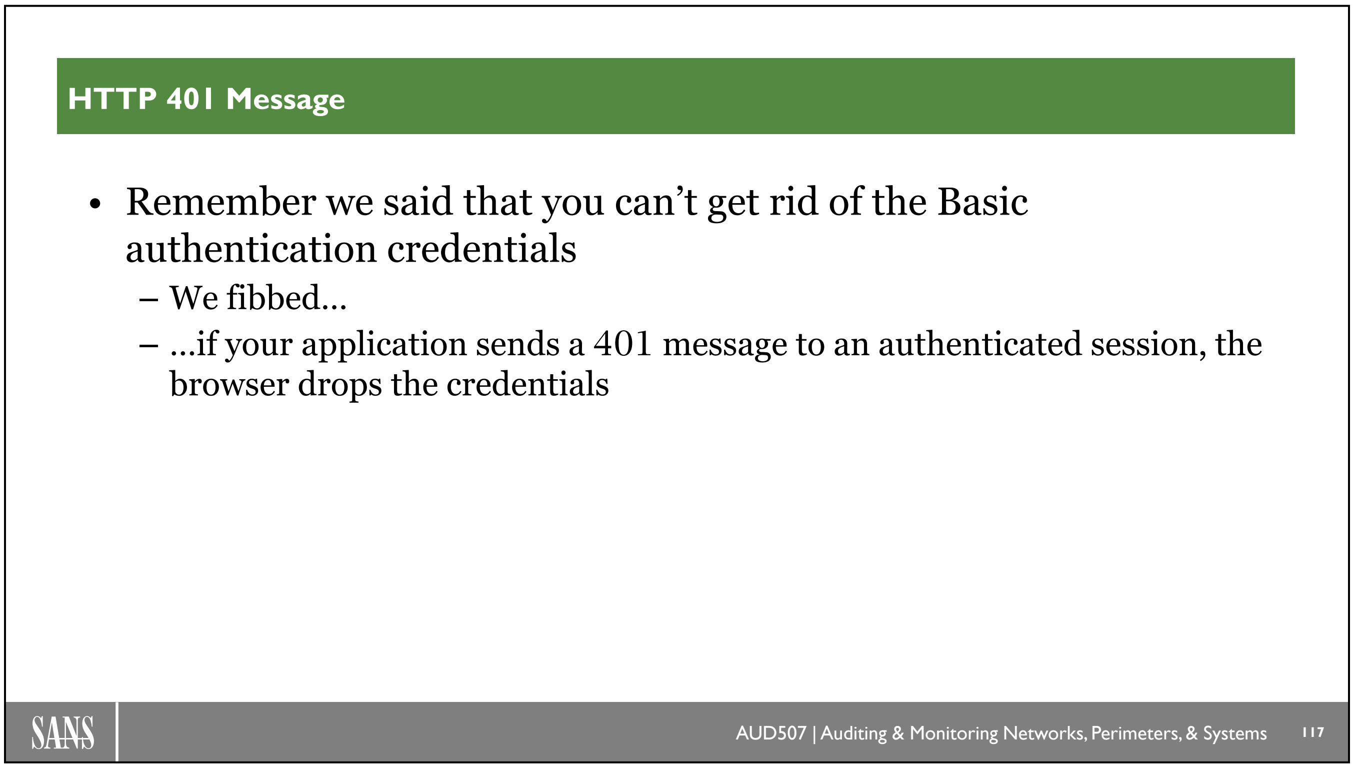
Just how easy is it to decode Basic authentication? There are websites that will allow you to paste in base64-encoded text and decode it. You can, of course, use Burp to do this as well. You will try this out for yourself in the next exercise!

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



HTTP 401 Message

- Remember we said that you can't get rid of the Basic authentication credentials
  - We fibbed...
  - ...if your application sends a 401 message to an authenticated session, the browser drops the credentials

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 117

When we spoke about Basic authentication, we made a point of explaining that the credentials are cached in the browser and that there's no way to get rid of those credentials short of closing the browser completely. As it turns out, we fibbed a bit. It is possible to flush the credentials.

The browser relies on the fact that the server continues to interact with it to decide if the credentials are valid. This is based entirely on the HTTP message codes. For instance, if the browser sends a set of credentials to a server and the server answers with an HTTP 200, the credentials must be correct (or unnecessary). However, if the server sends back an HTTP 401 message, that means that the credentials, if there were any, are incorrect.

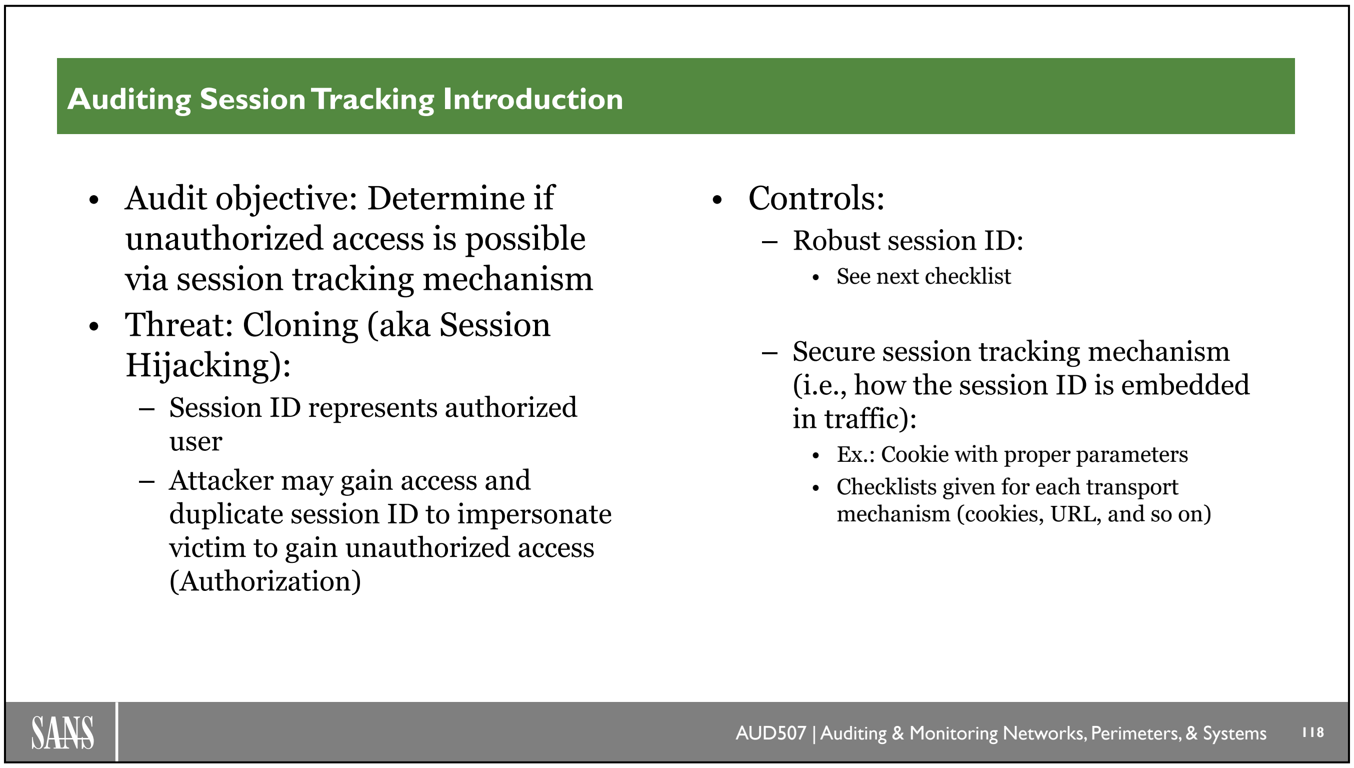
If the browser receives an HTTP 401 message from the server, it immediately drops the credentials because it assumes that they are invalid!

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Auditing Session Tracking Introduction

- Audit objective: Determine if unauthorized access is possible via session tracking mechanism
- Threat: Cloning (aka Session Hijacking):
  - Session ID represents authorized user
  - Attacker may gain access and duplicate session ID to impersonate victim to gain unauthorized access (Authorization)
- Controls:
  - Robust session ID:
    - See next checklist
  - Secure session tracking mechanism (i.e., how the session ID is embedded in traffic):
    - Ex.: Cookie with proper parameters
    - Checklists given for each transport mechanism (cookies, URL, and so on)

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 118

Now that we understand the session tracking mechanism, what we need to do is to determine if unauthorized access is possible by attacking the session ID or mucking around with the session tracking mechanism.

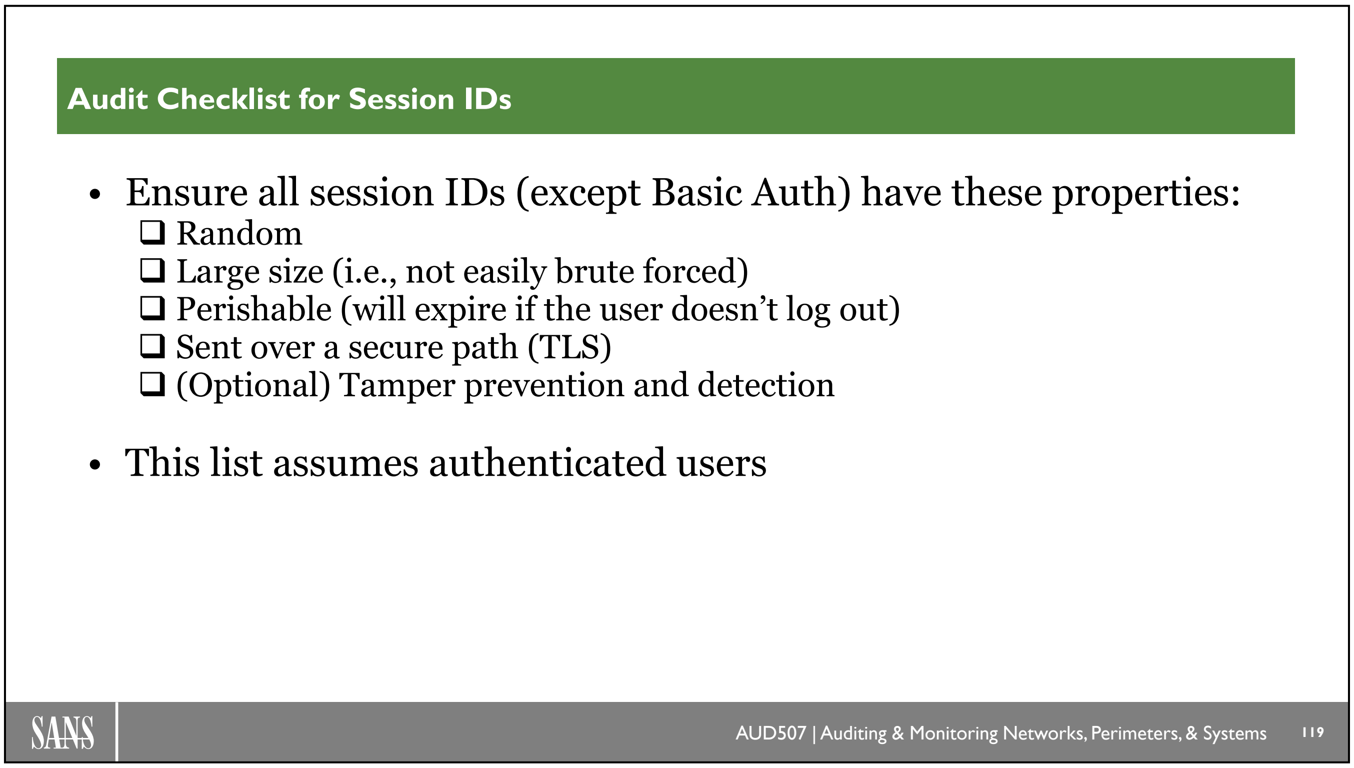
The threat of an attacker taking over someone else's application session by using the same session ID is called *session hijacking* in most publications. The term session hijacking is already used to describe when someone hijacks network traffic connections (that is, uses a TCP/IP sequence number prediction to take over your TCP/IP session) – for example, hijacking a telnet session. When that happens, victims can typically tell something is happening because they suddenly don't see any responses from the remote server. For example, characters they type, which are normally echoed back to their terminal, are absent.

However, when someone steals or guesses your session ID while interacting with a web application, there will now be two of you interacting with the web application.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Audit Checklist for Session IDs

- Ensure all session IDs (except Basic Auth) have these properties:
  - ❑ Random
  - ❑ Large size (i.e., not easily brute forced)
  - ❑ Perishable (will expire if the user doesn't log out)
  - ❑ Sent over a secure path (TLS)
  - ❑ (Optional) Tamper prevention and detection
- This list assumes authenticated users

Regardless of whether you create your own session tracking mechanism or rely on someone else's, you should audit *both* the session tracking method and the session IDs used.

Randomness

The session ID must be as random as possible. If this ID is not random, there are several viable attacks against the ID (more later).

Perishable

Eventually expires and cannot be reused/replayed (short-term versus long-term secrets).

Secure Transport

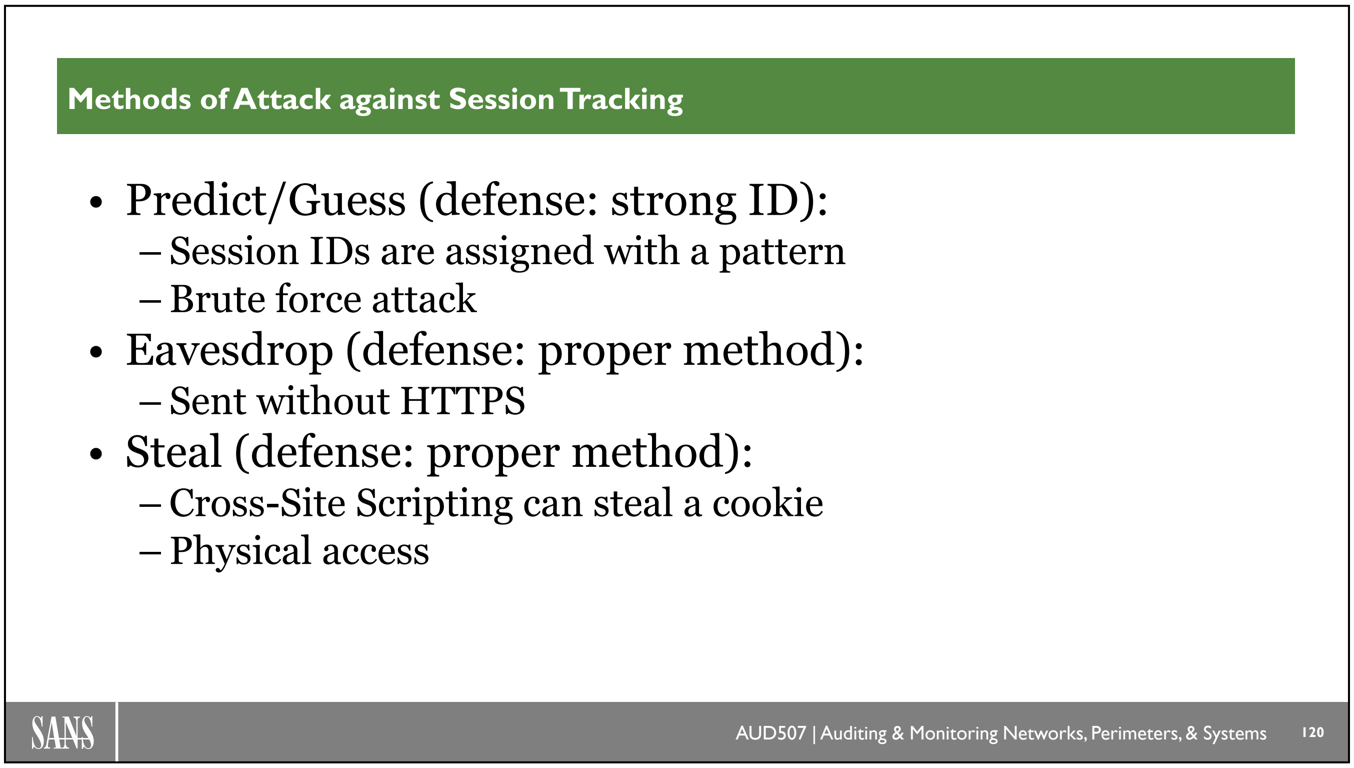
Sent over a secure path to prevent eavesdropping.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Methods of Attack against Session Tracking

- Predict/Guess (defense: strong ID):
  - Session IDs are assigned with a pattern
  - Brute force attack
- Eavesdrop (defense: proper method):
  - Sent without HTTPS
- Steal (defense: proper method):
  - Cross-Site Scripting can steal a cookie
  - Physical access

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 120

Now that we have covered the session tracking methods, let's focus more on how session cloning/hijacking takes place. The key to session cloning is the ability of a third party to use the same session ID as another user.

The threat of session cloning can be leveraged by one or more of the following attacks: Guessing, predicting, eavesdropping, or stealing the session ID.

Some of these threats can be mitigated with the proper use of the session tracking method employed, which we already covered. However, some of these threats can be mitigated only with the use of proper session IDs.

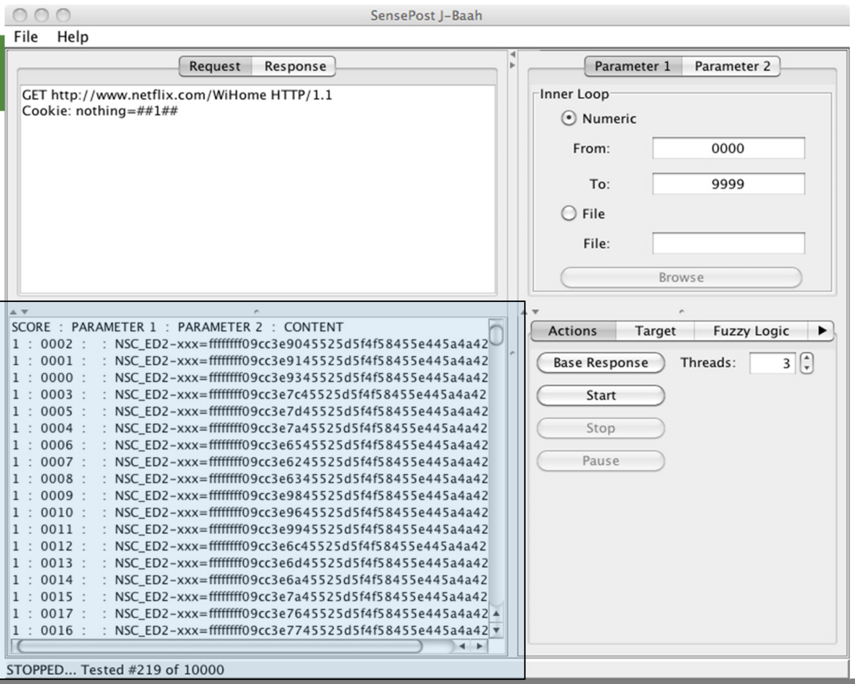
August 10, 2021

August 10, 2021

© SANS Institute 2021

Capturing Session IDs

- Netflix example
  - J-Baah used to gather a large number of session IDs
    - Are they long
    - Are they random
    - Can we predict what might come next



```
GET http://www.netflix.com/WiHome HTTP/1.1
Cookie: nothing=##1##
```

From: 0000
To: 9999
Threads: 3

```
SCORE  : PARAMETER 1  : PARAMETER 2  : CONTENT
1 : 0002 :   : NSC_ED2-xxx=ffffffff09cc3e9045525d5f4f58455e445a4a42
1 : 0001 :   : NSC_ED2-xxx=ffffffff09cc3e9145525d5f4f58455e445a4a42
1 : 0000 :   : NSC_ED2-xxx=ffffffff09cc3e9345525d5f4f58455e445a4a42
1 : 0003 :   : NSC_ED2-xxx=ffffffff09cc3e7c45525d5f4f58455e445a4a42
1 : 0005 :   : NSC_ED2-xxx=ffffffff09cc3e7d45525d5f4f58455e445a4a42
1 : 0004 :   : NSC_ED2-xxx=ffffffff09cc3e7a45525d5f4f58455e445a4a42
1 : 0006 :   : NSC_ED2-xxx=ffffffff09cc3e6545525d5f4f58455e445a4a42
1 : 0007 :   : NSC_ED2-xxx=ffffffff09cc3e6245525d5f4f58455e445a4a42
1 : 0008 :   : NSC_ED2-xxx=ffffffff09cc3e6345525d5f4f58455e445a4a42
1 : 0009 :   : NSC_ED2-xxx=ffffffff09cc3e9845525d5f4f58455e445a4a42
1 : 0010 :   : NSC_ED2-xxx=ffffffff09cc3e9645525d5f4f58455e445a4a42
1 : 0011 :   : NSC_ED2-xxx=ffffffff09cc3e9945525d5f4f58455e445a4a42
1 : 0012 :   : NSC_ED2-xxx=ffffffff09cc3e6c45525d5f4f58455e445a4a42
1 : 0013 :   : NSC_ED2-xxx=ffffffff09cc3e6d45525d5f4f58455e445a4a42
1 : 0014 :   : NSC_ED2-xxx=ffffffff09cc3e6a45525d5f4f58455e445a4a42
1 : 0015 :   : NSC_ED2-xxx=ffffffff09cc3e7a45525d5f4f58455e445a4a42
1 : 0017 :   : NSC_ED2-xxx=ffffffff09cc3e7645525d5f4f58455e445a4a42
1 : 0016 :   : NSC_ED2-xxx=ffffffff09cc3e7745525d5f4f58455e445a4a42
STOPPED... Tested #219 of 10000
```

First, you need to determine where the session ID is located within the traffic and when they are first sent. Typically, session IDs are within URLs, cookies, or hidden fields. This can be determined by examining your session with WebScarab. After logging all your web traffic, analyze it to determine where the session ID is located in the traffic.

An alternative is to use something such as J-Baah, which can't do the analysis, but for a fast look for easy-to-spot patterns, it's great!

To use it in this way, notice what we've done. We've set the target to point to the target application and used the Base Response button to experiment and find a request that generates a session ID. In this case, there appear to be two related session IDs: NetflixSession and nflxsid. With that information in hand, we use the Start and End token buttons to select the session ID we're interested in and press the Start button. That's it!

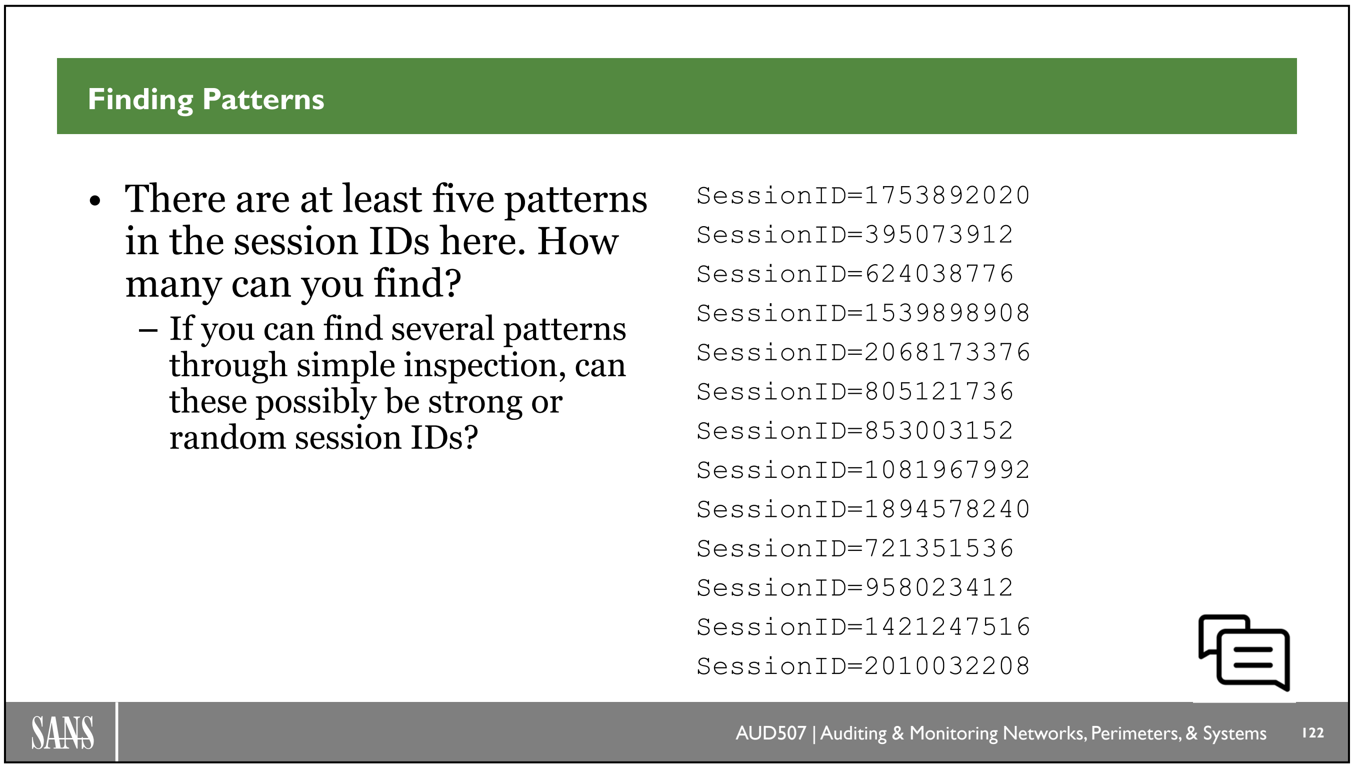
We can now let it run for a few seconds so that we can look for obvious patterns in the results.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Finding Patterns

- There are at least five patterns in the session IDs here. How many can you find?
  - If you can find several patterns through simple inspection, can these possibly be strong or random session IDs?

```
SessionID=1753892020
SessionID=395073912
SessionID=624038776
SessionID=1539898908
SessionID=2068173376
SessionID=805121736
SessionID=853003152
SessionID=1081967992
SessionID=1894578240
SessionID=721351536
SessionID=958023412
SessionID=1421247516
SessionID=2010032208
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 122

For a quick example of what we're talking about, look at the session IDs above that were captured from a web application. There are at least five distinct patterns that are readily apparent.

We have not included the answers here. Your instructor should help you to find at least five during class! If you are not at a live conference and want some possible answers, you can send an email to clay@risenhooverconsulting.com for some suggestions!

August 10, 2021

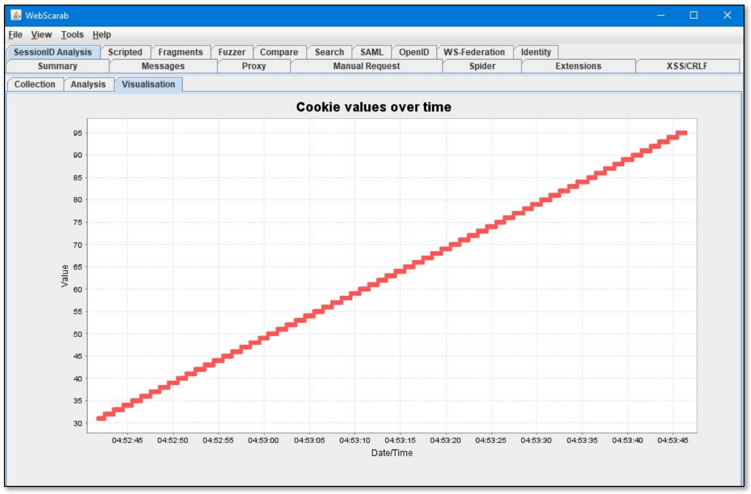
© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Graphing Session ID – WebScarab

- Webscarab has a really nice tool for this
- Graph represents session ID cookie values over time
- Does this graph look random?



The human eye is very good at detecting patterns which are NOT random. The straight diagonal line in this graph is a good example of something that is not random. Rather, the values used for session IDs in this application seem to increment predictably over time. It will not surprise you to learn that, in this application, session IDs are generated based on the time the user authenticated.

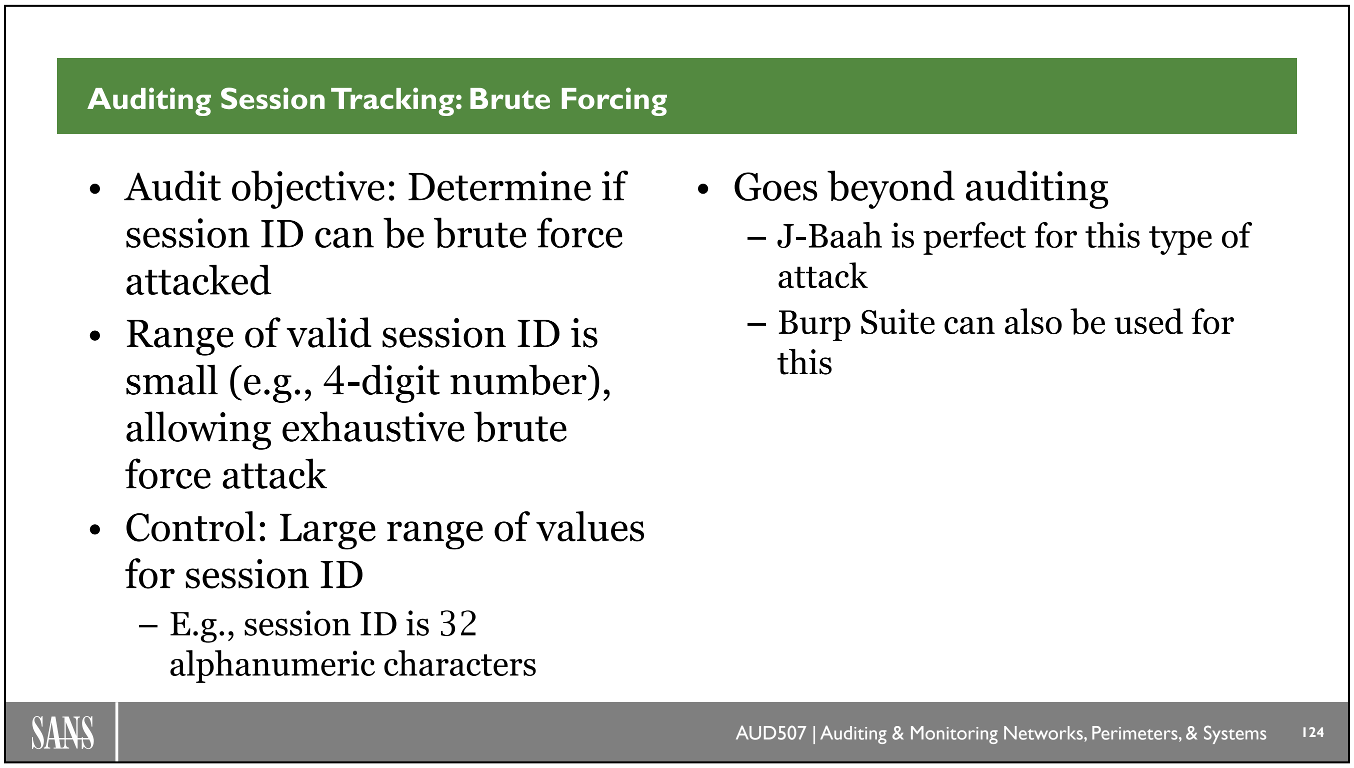
WebScarab is an older proxy written by the folks at OWASP (it predates ZAP). It is still part of my web application testing toolkit because of its session ID analysis tool. I find these graphs to be very compelling in reports. You can use Burp's tools to generate a much more sophisticated analysis of session ID randomness, but I find it to be harder to explain to those reading my audit report.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Auditing Session Tracking: Brute Forcing

- Audit objective: Determine if session ID can be brute force attacked
- Range of valid session ID is small (e.g., 4-digit number), allowing exhaustive brute force attack
- Control: Large range of values for session ID
  - E.g., session ID is 32 alphanumeric characters
- Goes beyond auditing
  - J-Baah is perfect for this type of attack
  - Burp Suite can also be used for this

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 124

If a session ID cannot be predicted, then perhaps it can be brute forced. This is where a person would simply request every possible session ID value within a range.

To perform this type of attack, we are essentially doing exactly what we do in a brute force password attack except that there is no username! In many ways, this attack is actually easier because there is almost never any kind of lockout in place for a large number of bad requests from one host with multiple session IDs. Another reason is that we don't have to come up with a valid username, just a valid session ID! This is the reason that we need a reasonably long session ID value. A minimum of 40 to 64 bytes should be sufficient today provided the value is significantly random and has a large key space. (The key spaces are the characters actually used to make up the session ID.)

J-Baah, it turns out, is excellent for this type of attack. We can select individual values and cycle through them automatically, sending request after request.

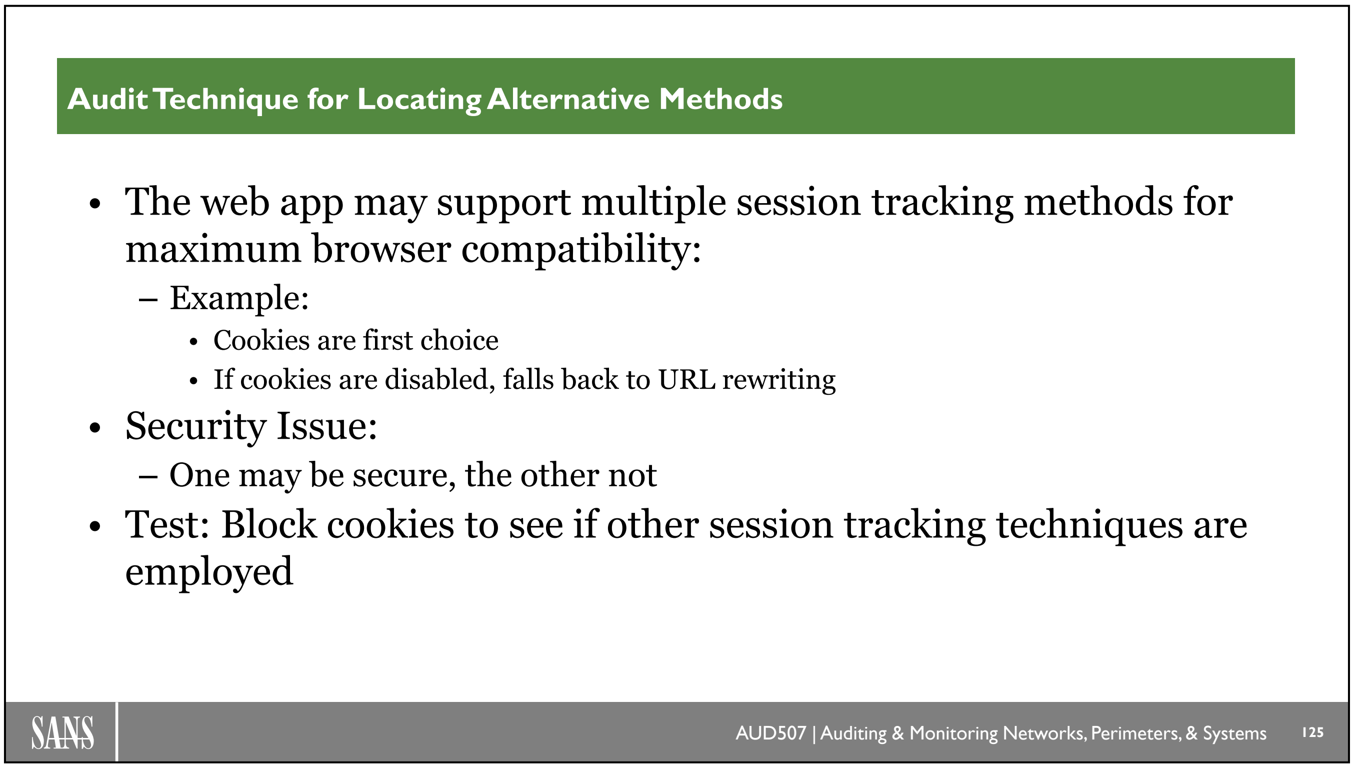
Burp Suite's Intruder tool is also easy to use to perform this type of attack.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Audit Technique for Locating Alternative Methods

- The web app may support multiple session tracking methods for maximum browser compatibility:
  - Example:
    - Cookies are first choice
    - If cookies are disabled, falls back to URL rewriting
- Security Issue:
  - One may be secure, the other not
- Test: Block cookies to see if other session tracking techniques are employed

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 125

Before using cookies to track a session, the web app may test for cookie acceptance from the user's browser. If cookies are allowed, then the web app places the session tracking ID inside a cookie. But what if you intentionally turn off cookies? The web app may support alternative forms of session tracking, such as URL rewriting.

The cookies may be a secure form of session tracking, but the URL rewriting may not be secure (for example placed on the end of non-encrypted URLs).

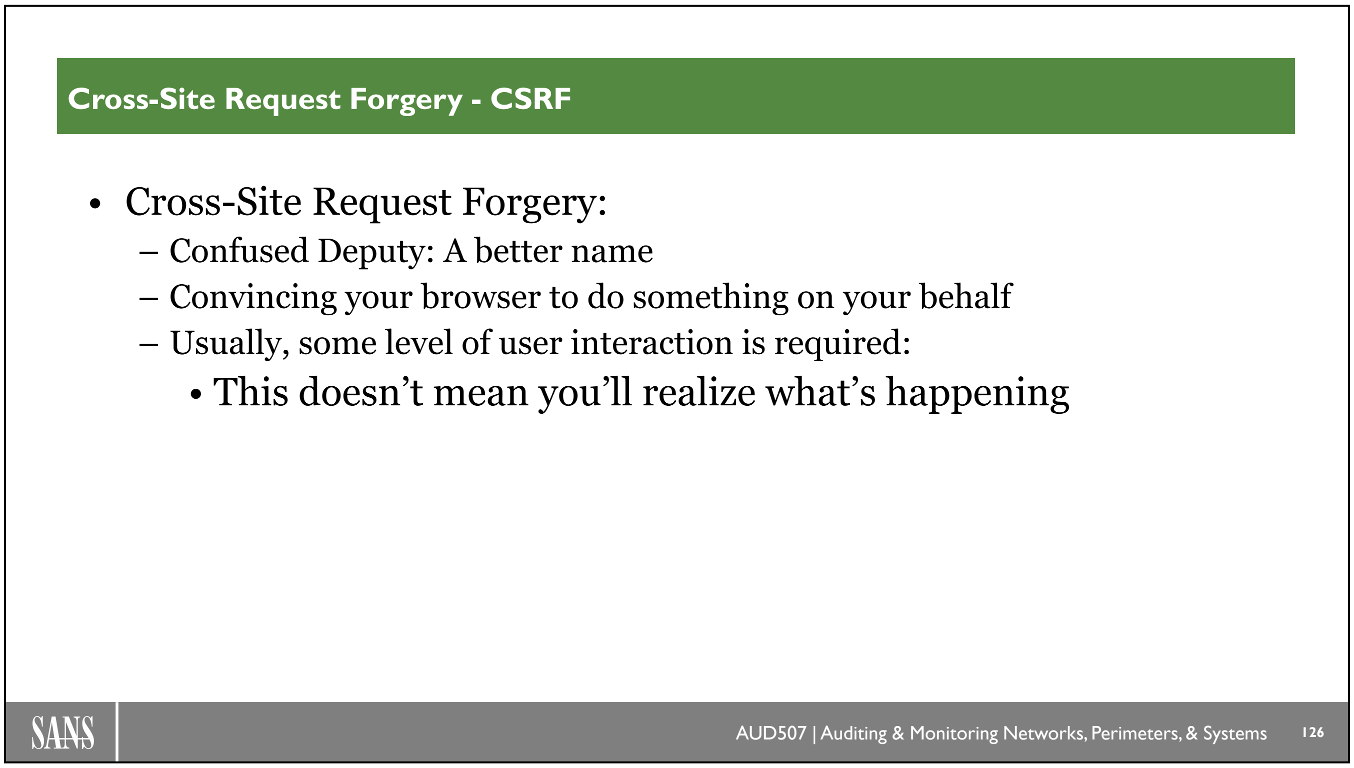
It's a good idea to test cookies (if they are used) and to then block cookies to see if other session tracking methods are employed. If other methods are used, then analyze them for the issues we just covered in this section.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Cross-Site Request Forgery - CSRF

- Cross-Site Request Forgery:
  - Confused Deputy: A better name
  - Convincing your browser to do something on your behalf
  - Usually, some level of user interaction is required:
    - This doesn't mean you'll realize what's happening

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 126

Cross-Site Request Forgery is a problem that has existed for a long time but has only recently been getting a great deal of attention. I think that one of the reasons that it hasn't had much attention until now is that we had so many things that were easier to exploit! Another is that fixing this problem isn't as easy as fixing most of the other problems that we have.

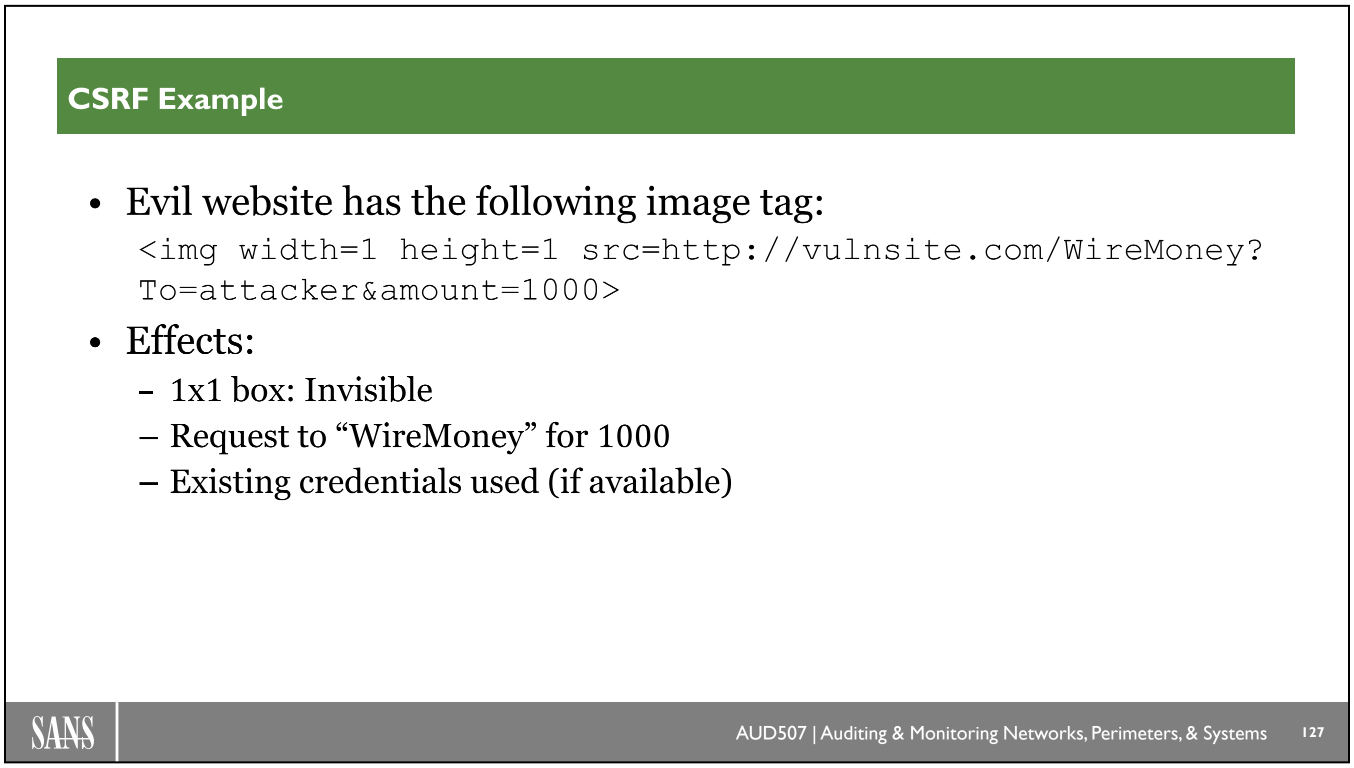
The basic idea is that an attacker embeds code onto one webpage that convinces your browser to go to another site and take some action on your behalf. For this to work the user being attacked must have valid credentials on the target site, and the user must be currently authenticated or have credentials in the form of a cookie (or possibly have the username and password stored in a password cache). Of course, there has to be something worth attacking, too, but that's usually a low threshold to overcome.

What's so interesting about this attack is that the person being compromised may have no indication that it's happening!

August 10, 2021

August 10, 2021

© SANS Institute 2021



CSRF Example

- Evil website has the following image tag:
```
<img width=1 height=1 src=http://vulnsite.com/WireMoney?
To=attacker&amount=1000>
```
- Effects:
  - 1x1 box: Invisible
  - Request to "WireMoney" for 1000
  - Existing credentials used (if available)

As a simple example, the original CSRF problem that was discussed in 1998 had to do with an image tag referencing something that wasn't an image. (CSRF was actually first discussed in 1998 as a "Confused Deputy" problem).

In our example, the image tag, which would normally point to an image or something that would produce an image, is instead referring to another web application. In this case, we're using an imaginary wire transfer application that takes a destination account number and an amount of money to transfer. Notice that the source URL is a properly formatted URL for that fictional application. Also, notice that the image is set to a width and height of 1 pixel. The effect of this will be a single dot on our browser window that will look like a screen artifact or a period. Notably, it will *not* look like a broken image tag!

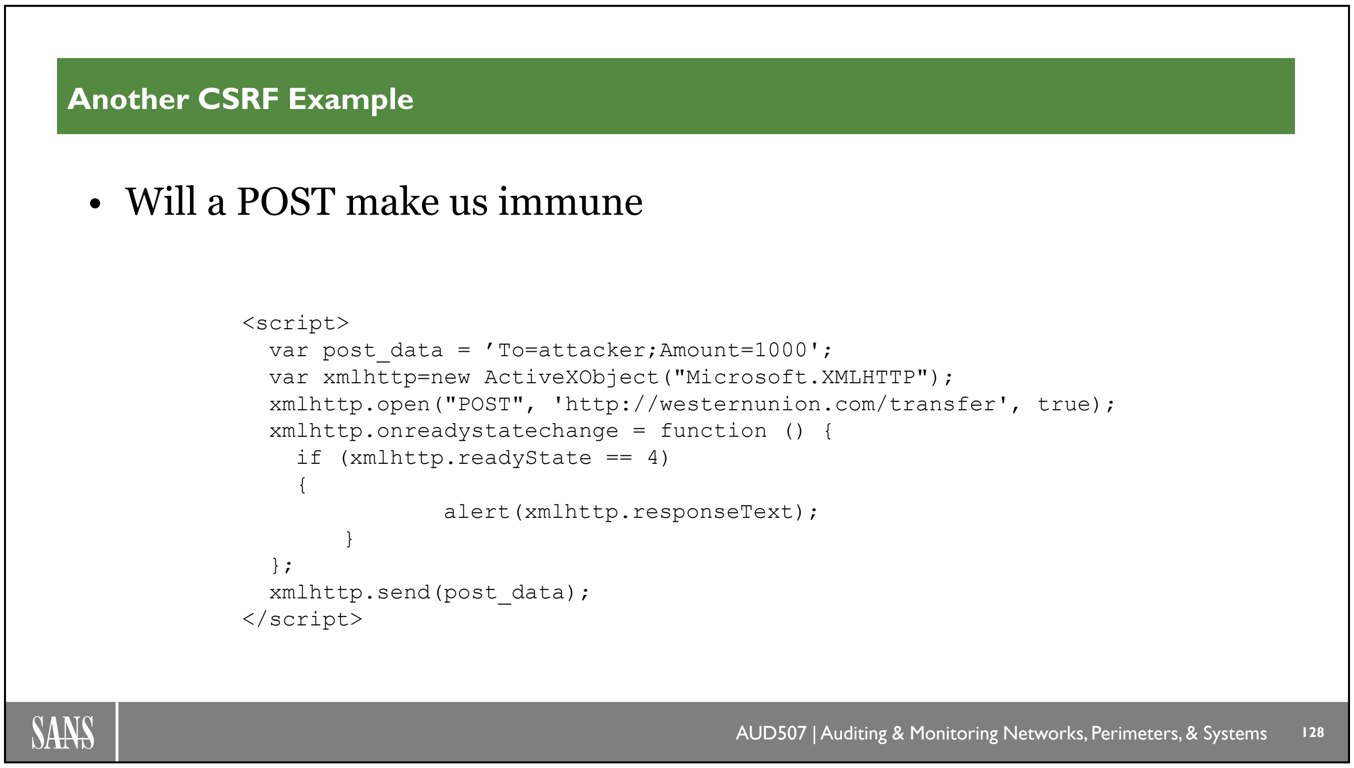
For this to work, remember that the user who is browsing the site would need existing credentials stored in a cookie or in the browser. If the user doesn't have credentials, there won't be a transfer, but there also won't be an error. The beauty of this attack is that embedding this image tag someplace such as eBay could result in hundreds of thousands of hits. The more hits we get, the better the chance that we'll find someone who it works against. Targeted attacks will, of course, increase the success rate.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Another CSRF Example

- Will a POST make us immune

```
<script>
  var post_data = 'To=attacker;Amount=1000';
  var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  xmlhttp.open("POST", 'http://westernunion.com/transfer', true);
  xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4)
    {
               alert(xmlhttp.responseText);
        }
  };
  xmlhttp.send(post_data);
</script>
```

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 128

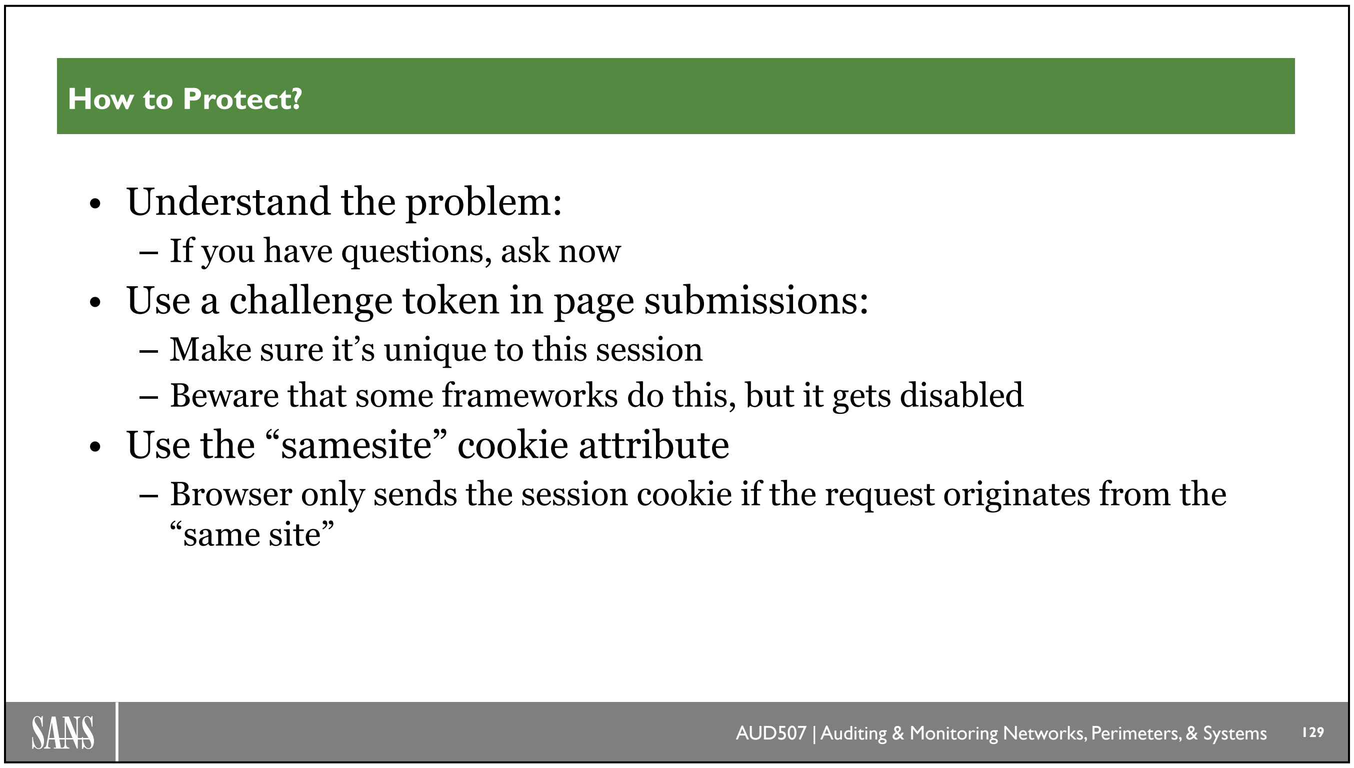
The most common myth that programmers believe about CSRF is that their applications are not affected because they are using POST for all their submissions. It is excellent that they are using a POST, but this does nothing to solve the problem. In this slide, we have included some JavaScript for Internet Explorer that could be embedded on a page, in an image tag, or virtually any other kind of tag. This JavaScript launches exactly the same attack as the previous slide, but this time uses a POST rather than a GET!

The details of how to create a CSRF attack are not important for us. What is important is understanding that simply changing an application to use a POST will not solve this problem. Of course, this script could easily be adapted to any application where a POST is required.

August 10, 2021

August 10, 2021

© SANS Institute 2021



How to Protect?

- Understand the problem:
  – If you have questions, ask now
- Use a challenge token in page submissions:
  – Make sure it's unique to this session
  – Beware that some frameworks do this, but it gets disabled
- Use the "samesite" cookie attribute
  – Browser only sends the session cookie if the request originates from the "same site"

The first step toward fixing this problem is understanding it. Most often I find that the programmers don't get what's happening here. If they don't understand and we don't understand, how can we possibly explain it to them? If you have questions about what this is and how it works, now is the time to ask!

When we understand the problem, we can write some code. One solution is to use some type of challenge-response token for sensitive page submissions. As a caution, I've seen instances in which the coders tried to do this and created a random challenge that could be verified, but it wasn't tied to the user session in any way. In other words, the attacker could go to your site, collect a valid token, insert it into another transaction, and successfully exploit the application! This means that the token must be tied to the current user session in some way, possibly by using the user ID, session ID, or some internal value related to the user as a key for the token.

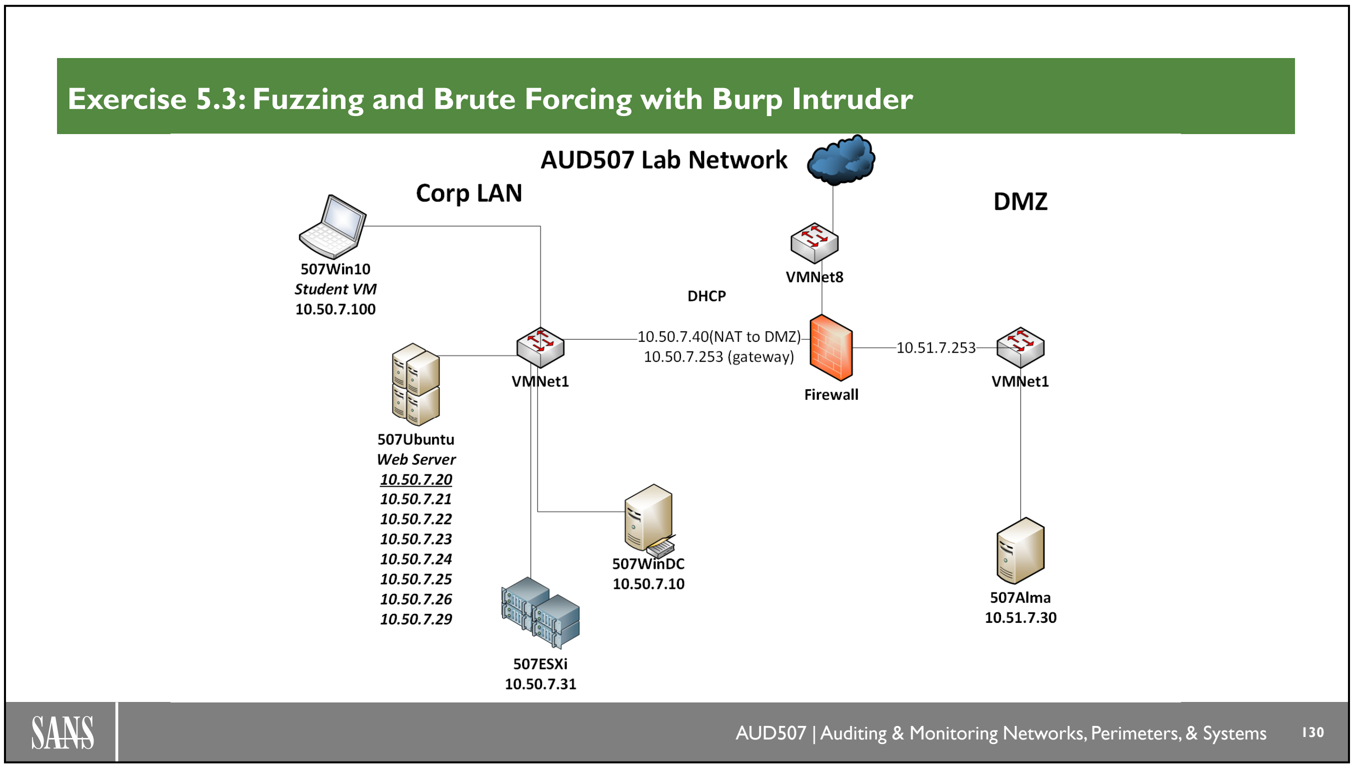
The best solution today is to use the "Samesite" cookie attribute. This attribute instructs the browser only to send the cookie back to the application if the request came from a page on the "same site" as the cookie's original origin.

August 10, 2021

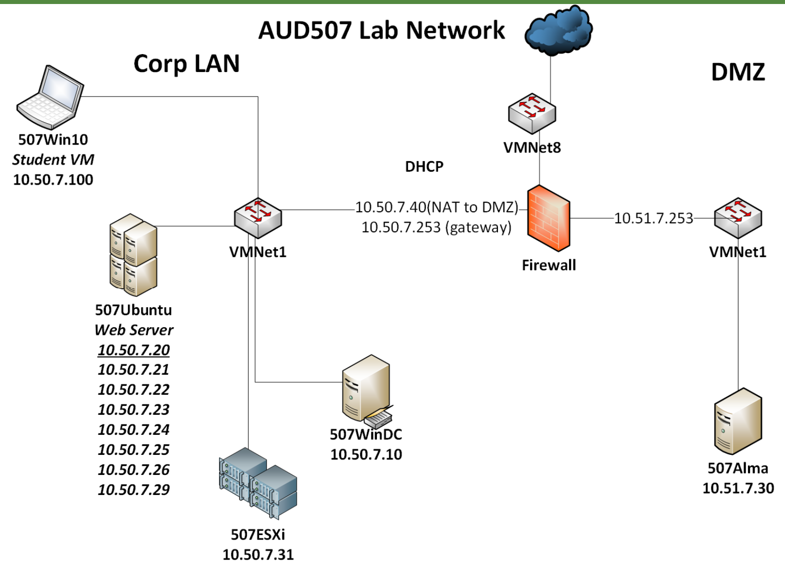
August 10, 2021

Technet24

© SANS Institute 2021



Exercise 5.3: Fuzzing and Brute Forcing with Burp Intruder

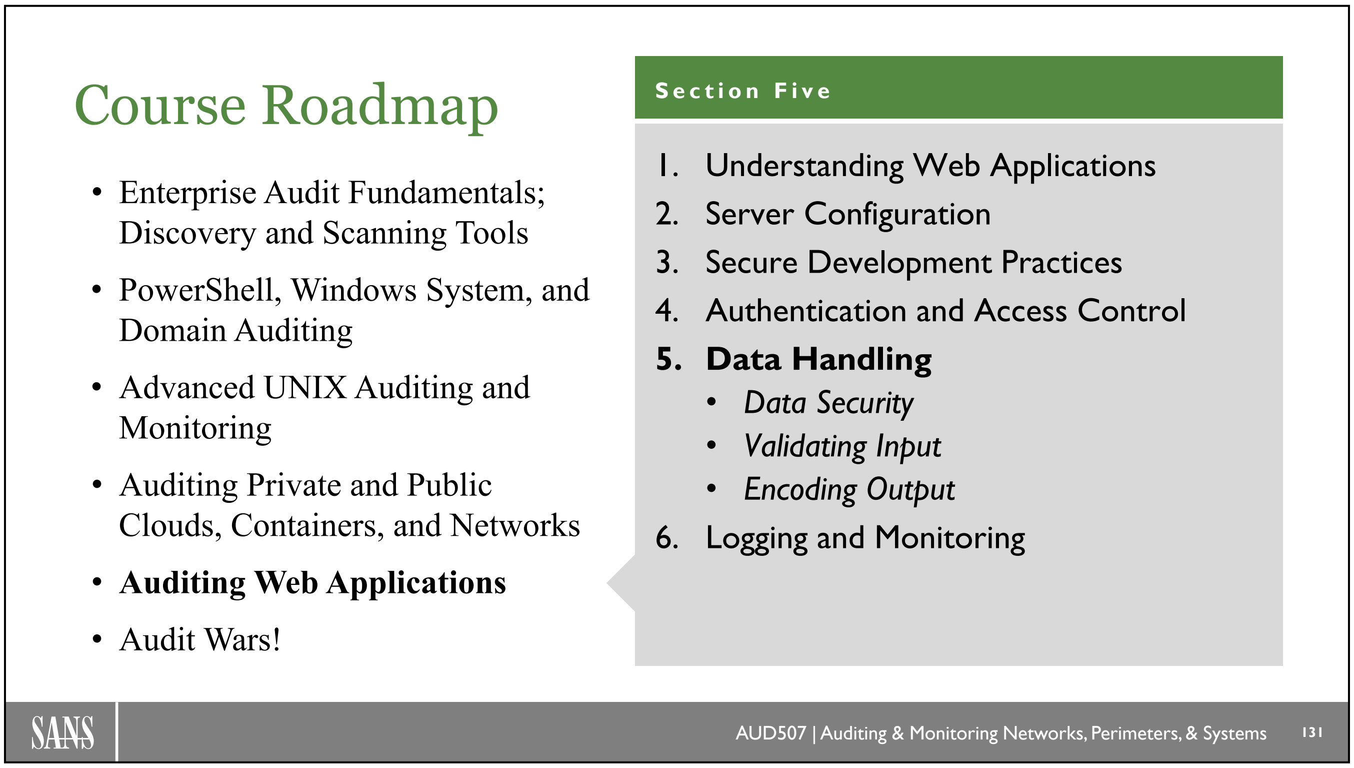


Please continue working in your workbook with Exercise 5.3: Fuzzing and Brute Forcing with Burp Intruder.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Course Roadmap

- Enterprise Audit Fundamentals; Discovery and Scanning Tools
- PowerShell, Windows System, and Domain Auditing
- Advanced UNIX Auditing and Monitoring
- Auditing Private and Public Clouds, Containers, and Networks
- **Auditing Web Applications**
- Audit Wars!

Section Five

1. Understanding Web Applications
2. Server Configuration
3. Secure Development Practices
4. Authentication and Access Control
5. **Data Handling**
   - *Data Security*
   - *Validating Input*
   - *Encoding Output*
6. Logging and Monitoring

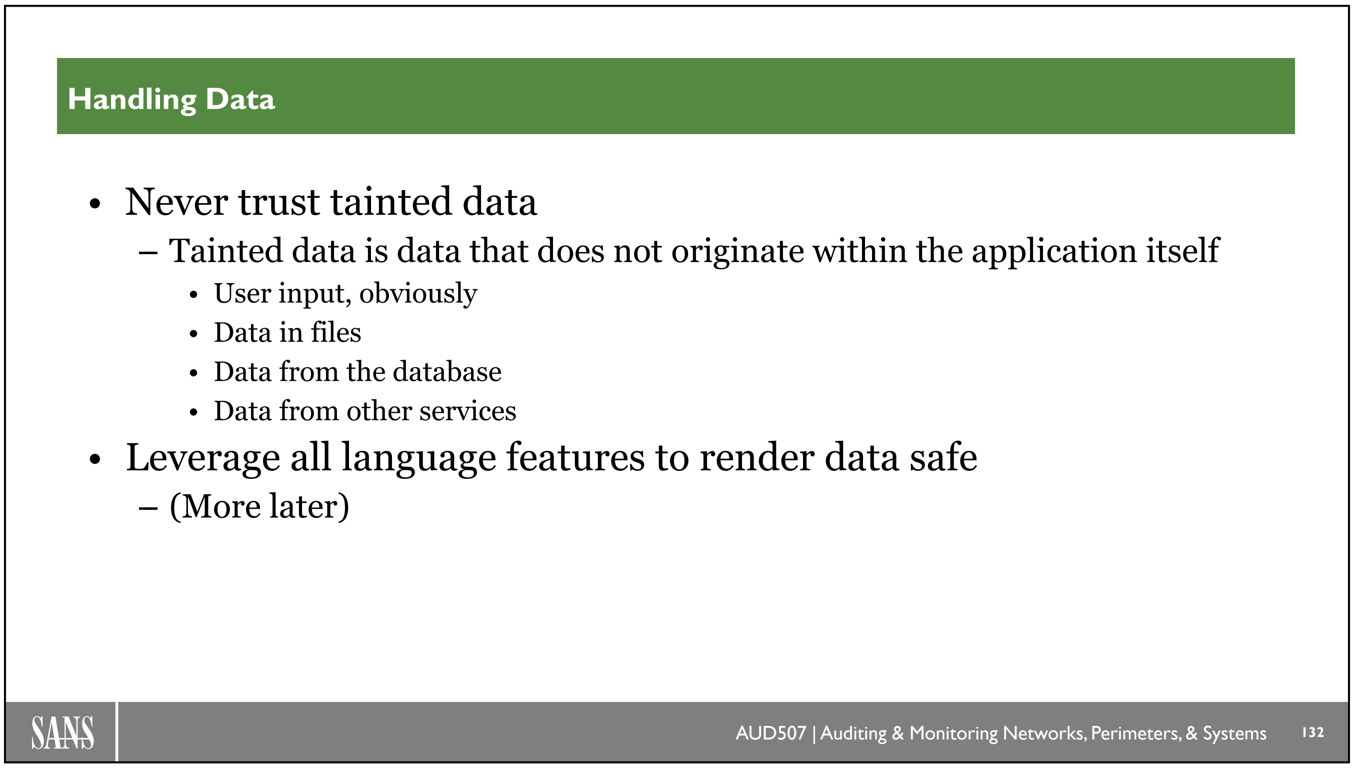
This page intentionally left blank.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Handling Data

- Never trust tainted data
  - Tainted data is data that does not originate within the application itself
    - User input, obviously
    - Data in files
    - Data from the database
    - Data from other services
- Leverage all language features to render data safe
  - (More later)

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 132

When it comes to the handling of data within the application, our organization should establish a standard that data is not trusted unless the application itself is the originator of the data. Programmers, when first hearing this, may think that this is more difficult. In fact, it isn't. This requirement simply calls for a different approach to the development process.

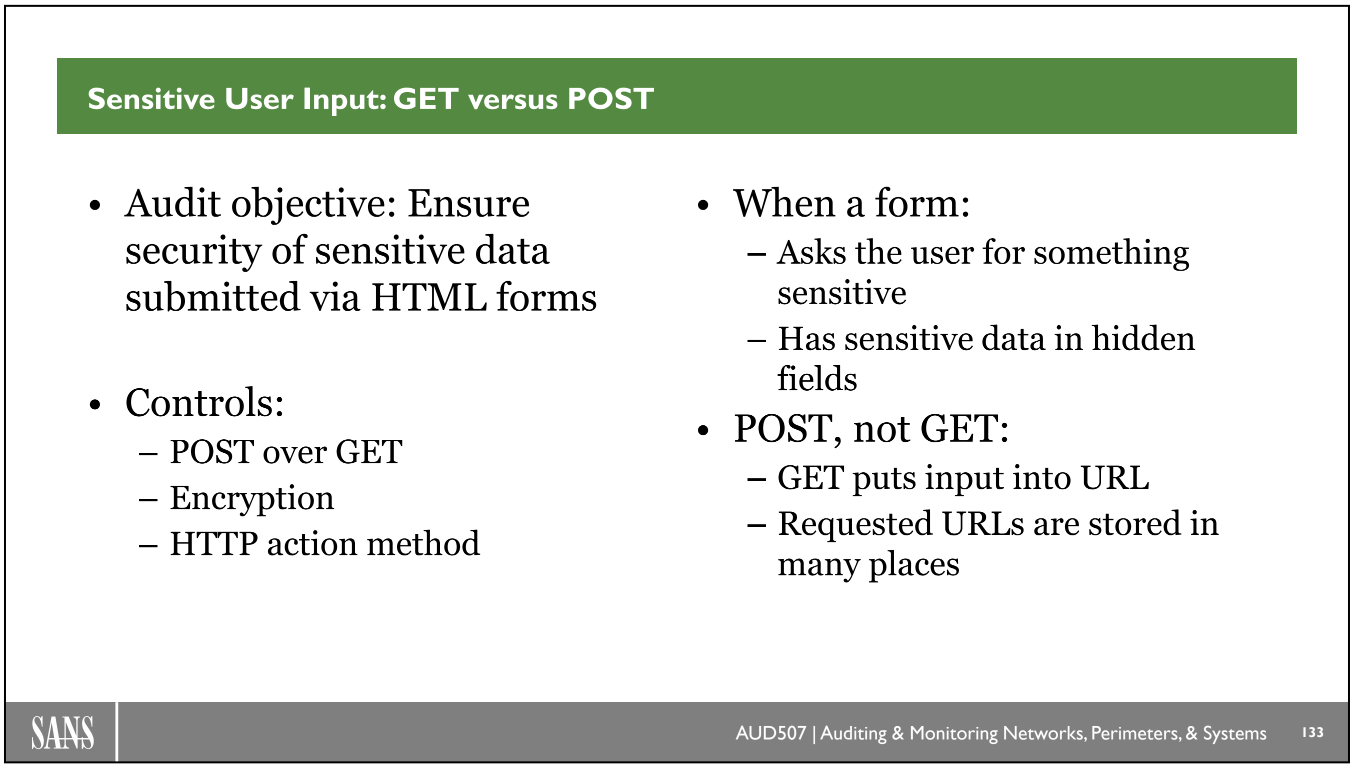
Others, when they hear this requirement, immediately think of user input. While this is certainly a source of tainted data, there are many other sources. Does the application *originate* data that is read out of the database, out of files on the file system, or obtained from other services? No! Therefore, none of these sources may be trusted either.

Developers can feel that this is going too far. Consider just one example. The data in the database seems as though it should be trustworthy. If, however, your application is actually useful to the enterprise, it is highly likely that other applications used internally in the enterprise make use of this data as well. These include shipping systems, customer service systems, billing systems, inventory systems, etc. Given that this is the case, there will clearly be multiple development teams creating applications that work with this data. How will this one application react when one of the other consumers of the data makes a modification that it does not expect? Could it lead to unexpected results? Could it even lead to a situation where data stored in one context is processed as code in another context?

We will examine the details of changing the context of data later when we cover injection flaws.

August 10, 2021

© SANS Institute 2021



Sensitive User Input: GET versus POST

- Audit objective: Ensure security of sensitive data submitted via HTML forms
- Controls:
  - POST over GET
  - Encryption
  - HTTP action method
- When a form:
  - Asks the user for something sensitive
  - Has sensitive data in hidden fields
- POST, not GET:
  - GET puts input into URL
  - Requested URLs are stored in many places

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 133

Let's look at this slide as a reminder. We already covered the GET and POST method briefly earlier in this section. We've come back to this topic once or twice, but now we will go into detail as to what the security issues of using one method instead of the other are.

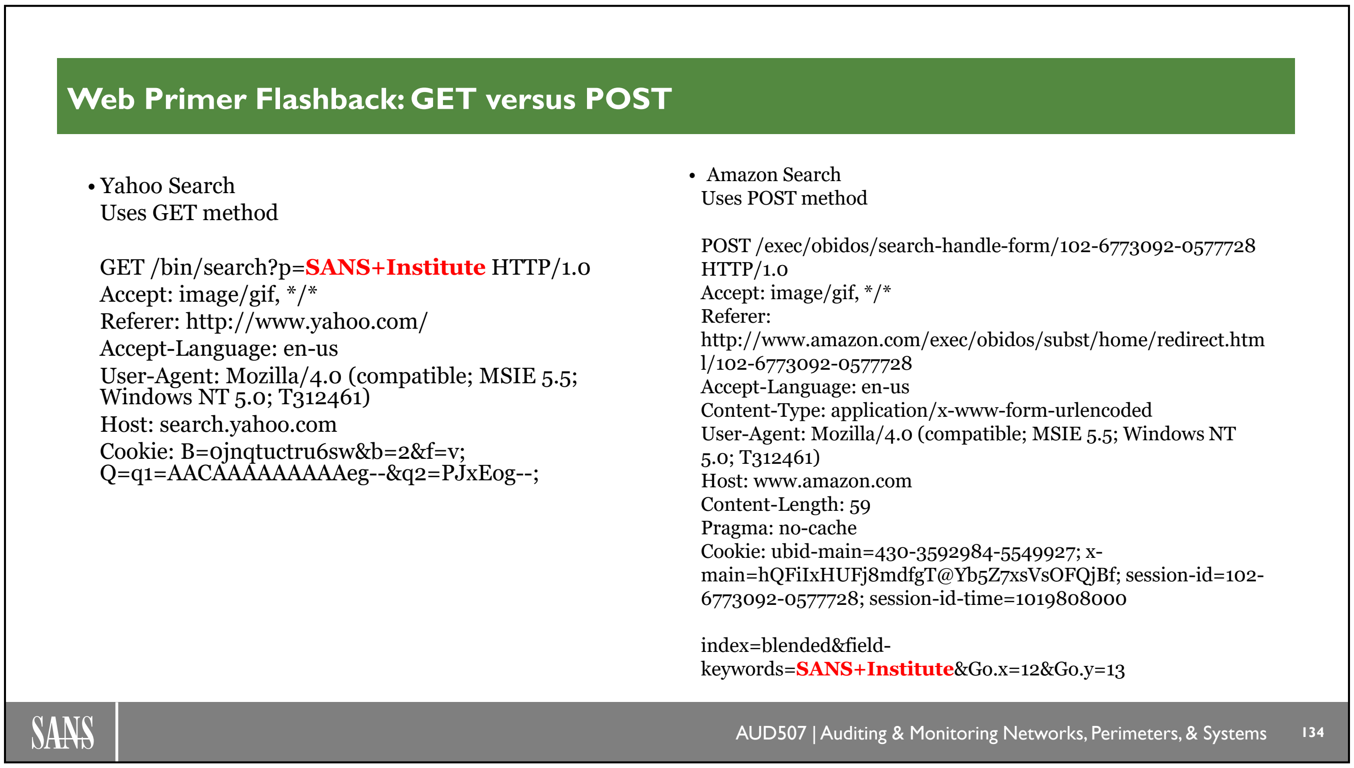
Our objective is to ensure that sensitive user data that is submitted via HTTP from HTML forms is done in a secure fashion. This entails the use of encryption as well as proper use of the HTTP action method (POST versus GET).

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Web Primer Flashback: GET versus POST

- Yahoo Search
  Uses GET method

```
GET /bin/search?p=SANS+Institute HTTP/1.0
Accept: image/gif, */*
Referer: http://www.yahoo.com/
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0; T312461)
Host: search.yahoo.com
Cookie: B=0jnqtuctru6sw&b=2&f=v;
Q=q1=AACAAAAAAAAAeg--&q2=PJxEog--;
```

- Amazon Search
  Uses POST method

```
POST /exec/obidos/search-handle-form/102-6773092-0577728
HTTP/1.0
Accept: image/gif, */*
Referer:
http://www.amazon.com/exec/obidos/subst/home/redirect.htm
l/102-6773092-0577728
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
5.0; T312461)
Host: www.amazon.com
Content-Length: 59
Pragma: no-cache
Cookie: ubid-main=430-3592984-5549927; x-
main=hQFiIxHUFj8mdfgT@Yb5Z7xsVsOFQjBf; session-id=102-
6773092-0577728; session-id-time=1019808000

index=blended&field-
keywords=SANS+Institute&Go.x=12&Go.y=13
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 134

On the left, we see the HTTP request sent from a browser submitting a search to Yahoo's search engine for the phrase "SANS Institute". Yahoo's search form uses the GET method. We see that the user input is placed **inside the URL being requested**.

On the right, we see the HTTP request sent from a browser submitting a search to Amazon search engine for the phrase "SANS Institute". Amazon.com's search form uses the POST method. We see that the user input is placed **inside the body of the request** being made.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Points of Exposure for GET Data

- The GET method is bad because it exposes user parameter values:
  - In the user's web browser history file
  - In the web server's audit logs
  - Intermediate proxy servers
  - At other websites via the HTTP “Referer” field

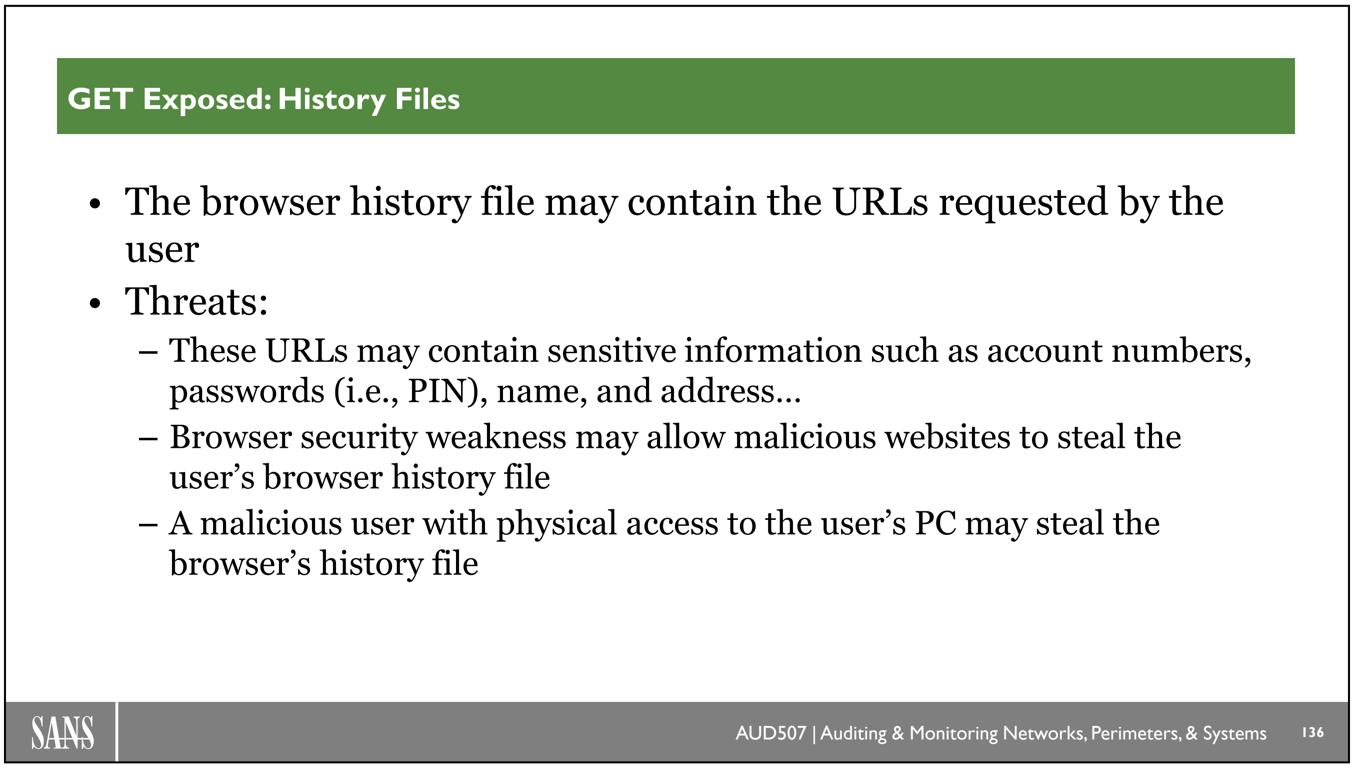
So, what does this mean in a nutshell? The GET method can be a significant point of exposure for sensitive data because it will be cached in several places. First, it is stored on the users’ local system in the browser history file. This can be viewed in plaintext on the hard drive or can be seen by pulling down the browser location bar. Next, this data is stored in the web server’s access logs. Third, the data might be stored in intervening proxy servers. Last, the data can be sent to third parties via the HTTP “Referer” field.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



GET Exposed: History Files

- The browser history file may contain the URLs requested by the user
- Threats:
  - These URLs may contain sensitive information such as account numbers, passwords (i.e., PIN), name, and address...
  - Browser security weakness may allow malicious websites to steal the user's browser history file
  - A malicious user with physical access to the user's PC may steal the browser's history file

When the URL is stored in the local history file of the user, we may have the feeling that this isn't particularly bad. The user may also feel that this isn't a particularly big issue. The fact is, though, it is quite serious when we consider what sort of information we might be talking about.

What if a GET request were used to submit a credit card number, Social Security number, bank account number, and so on. That information is now stored on the hard drive in an unencrypted form, which is a very bad idea. Worse, this type of behavior creates the motivation for individuals to discover browser flaws that allow them to remotely expose locally cached data, allowing an attacker to collect this sensitive information remotely.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

GET Exposed: History Files Example

Real Example:

https://www.BadBank.com/scripts/bankh.dll?Func=chmempro10503&homepath=cu3&LastName=CARLSON&FirstName=PAULA+B&**SSN=123456789&DOB=04%2F29%2F1948**&YrInSchool=0&HmFax=&Sex=F&CountryCd=&CountryDesc=&YrAtAddr=0&HouseType=&Marital=&NoDeps=0&DepAges=&**DrLNO=**&StateIssued=&**Addr1=7154+ST+RAYMOND+COURT**&Addr2=&**City=DUBLIN&State=CA&Zip=94568**&DayPhone=415+555-2135&EvePhone=415+555-2819&**mempwd=1234**

The above data dump is from another online banking site. At first, the site seemed well behaved in its use of HTTP request methods. Every transaction that used sensitive data used the POST method. Every transaction except one. It was one of the last tests performed during the audit. The request was to submit a change of address. Now that type of transaction is typically sensitive because an attacker could change your address and then order new checks, thus allowing him to intercept the checks without you knowing, especially if the new address is a vacant apartment. Given the security implications of this transaction, the site required the user to enter her password to submit a change of address.

The URL above was captured in the browser's history file. Can you imagine the PR nightmare if users noticed their Social Security number in their web history file?

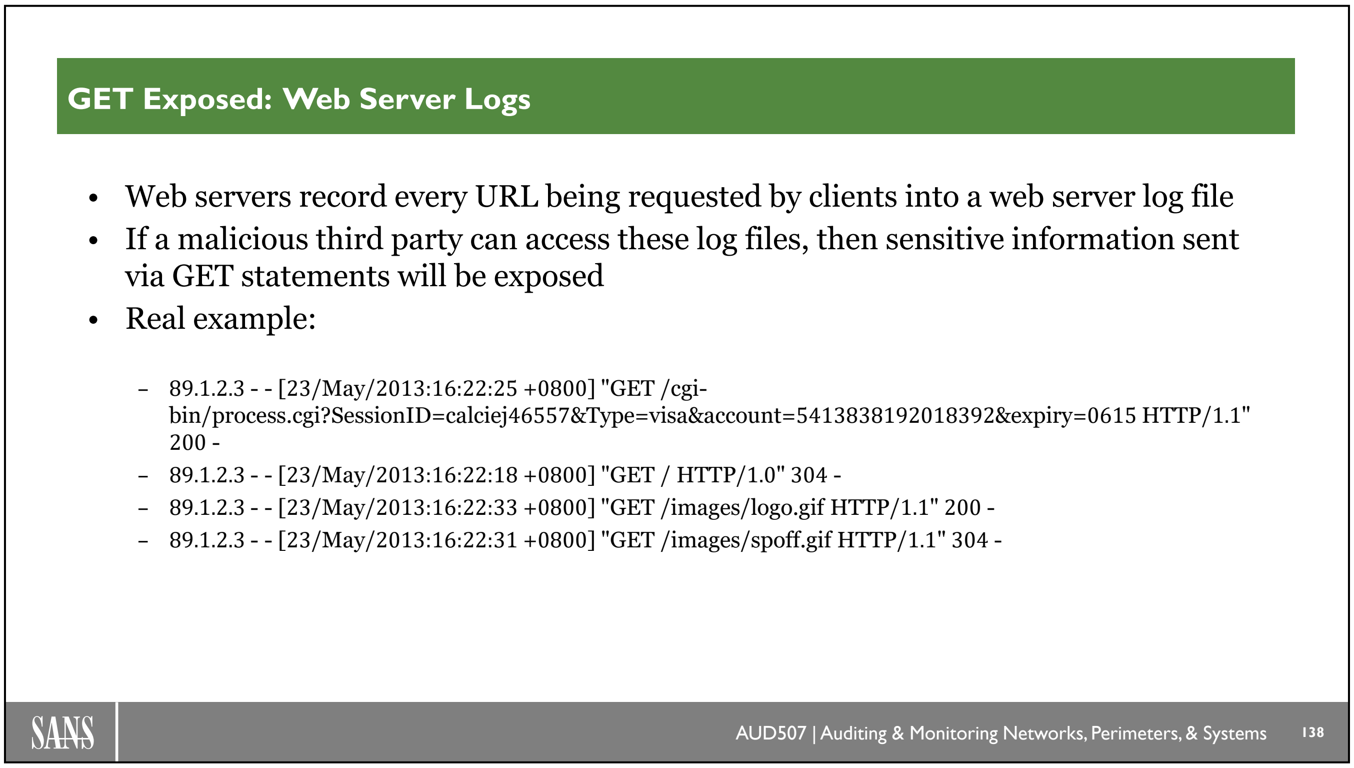
Where did this data come from? All I was asked to enter into the form was my new address and my password. All the other form elements were hidden form elements, thus submitted automatically for me. This is definitely one HTML page you don't want cached in cleartext on your PC.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



GET Exposed: Web Server Logs

- Web servers record every URL being requested by clients into a web server log file
- If a malicious third party can access these log files, then sensitive information sent via GET statements will be exposed
- Real example:

```
– 89.1.2.3 - - [23/May/2013:16:22:25 +0800] "GET /cgi-
  bin/process.cgi?SessionID=calciej46557&Type=visa&account=5413838192018392&expiry=0615 HTTP/1.1"
  200 -
– 89.1.2.3 - - [23/May/2013:16:22:18 +0800] "GET / HTTP/1.0" 304 -
– 89.1.2.3 - - [23/May/2013:16:22:33 +0800] "GET /images/logo.gif HTTP/1.1" 200 -
– 89.1.2.3 - - [23/May/2013:16:22:31 +0800] "GET /images/spoff.gif HTTP/1.1" 304 -
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 138

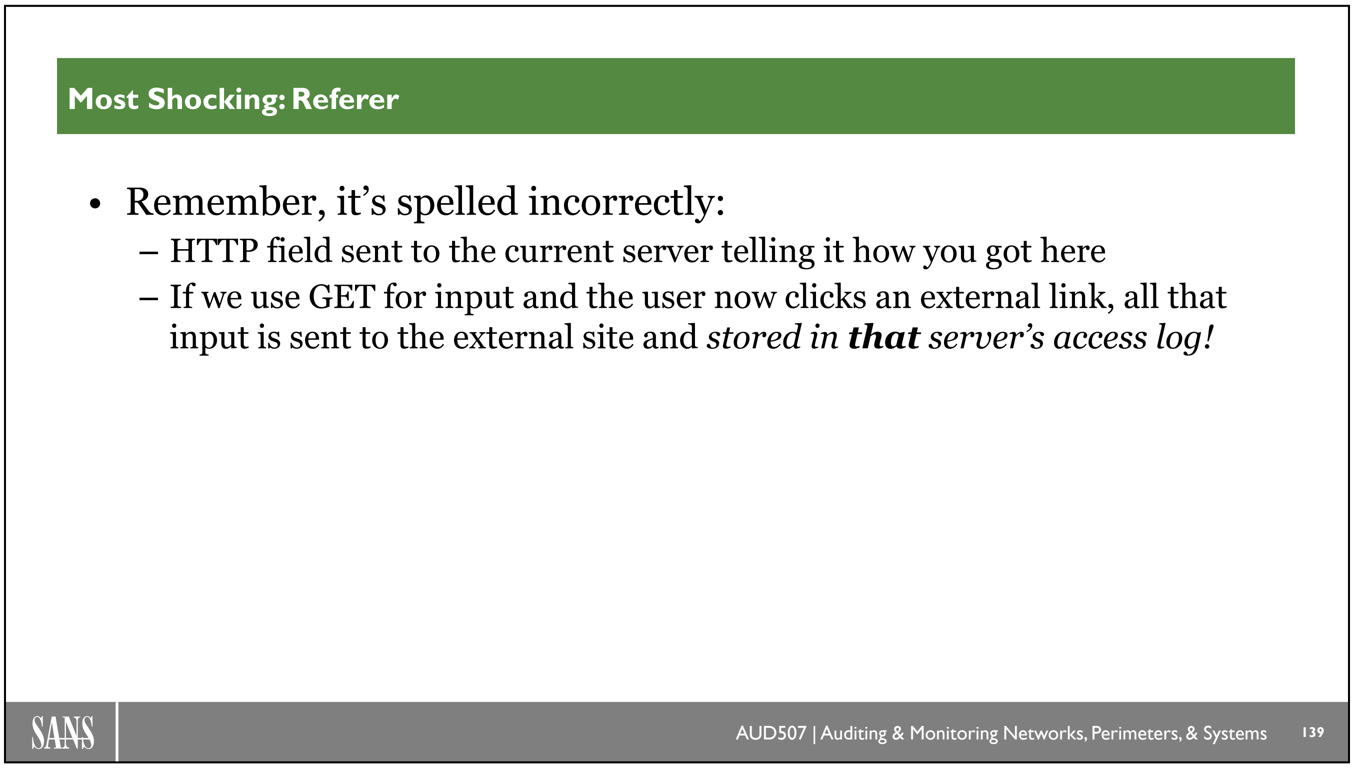
Real world example: The above web server logs were acquired remotely during a web application assessment by exploiting a web server weakness. Do not assume your web logs are safe. Even if they cannot be accessed remotely by unauthorized users, they can be seen by local web administrators who may not have a need to see the sensitive data embedded into the URLs.

It is also not terribly uncommon to find that through a misconfiguration the logs have been left in a public place or, if you pay to have your site hosted by a web hosting service, your web logs may be in a public place by default!

August 10, 2021

August 10, 2021

© SANS Institute 2021



Most Shocking: Referer

- Remember, it's spelled incorrectly:
  - HTTP field sent to the current server telling it how you got here
  - If we use GET for input and the user now clicks an external link, all that input is sent to the external site and *stored in **that** server's access log!*

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 139

If you're still not convinced, possibly the best reason to never use a GET for input is the referer field. We saw this HTTP header earlier. Recall that this field tells the server how we got here. The referer contains the entire URL including all parameters that were sent.

What's the impact? Imagine that we have an application that accepts input using a GET. Imagine that our user has just finished doing something that is somewhat sensitive and that the parameters for that last query are now sitting in the URL on the browser bar. The user, having finished what he planned to do, now clicks a Google result or clicks an advertisement that is also on the page. The browser now sends a request to this third party happily including the current URL in the referer field! Not only is the data not encrypted but it's now stored in plaintext on someone *else's* server!

(Please note, "Referer" looks as though it is spelled incorrectly. In fact, in the original standard, it was spelled incorrectly but this became ratified in the standard itself! When dealing with this field in HTTP, we must refer to it as the "Referer.")

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

User Inputs: Testing Scope

- What should be manipulated
  - All form elements
  - All cookies
  - Session ID
  - HTTP headers used by site/app

Now that we've been clear about why the GET method can be so dangerous for sensitive data, let's turn our attention to user input validation and testing. Remember we said that we want to try to put bad stuff into every opening we can find in the web application. This means every opening! It's easy to overlook some, so don't forget the cookies, the HTTP headers, and so on. What should you put into these openings? Let's look at that more closely.

August 10, 2021

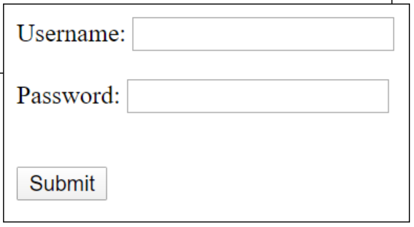
© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

SQL Injection - User Input Form

```
<html>
<head><title>SQL Inj Demo</title></head>
<body>
    <form action="login.aspx" method="post">
        Username: <input type=text name=username><br /><br />
        Password: <input type=password name=pwd></form><br />
        <input type=submit name=logMein>
    </form>
</body>
</html>
```



To help us with our discussion of SQL injection in web applications, we have designed a very simple login page to be used as part of a form-based authentication scheme.

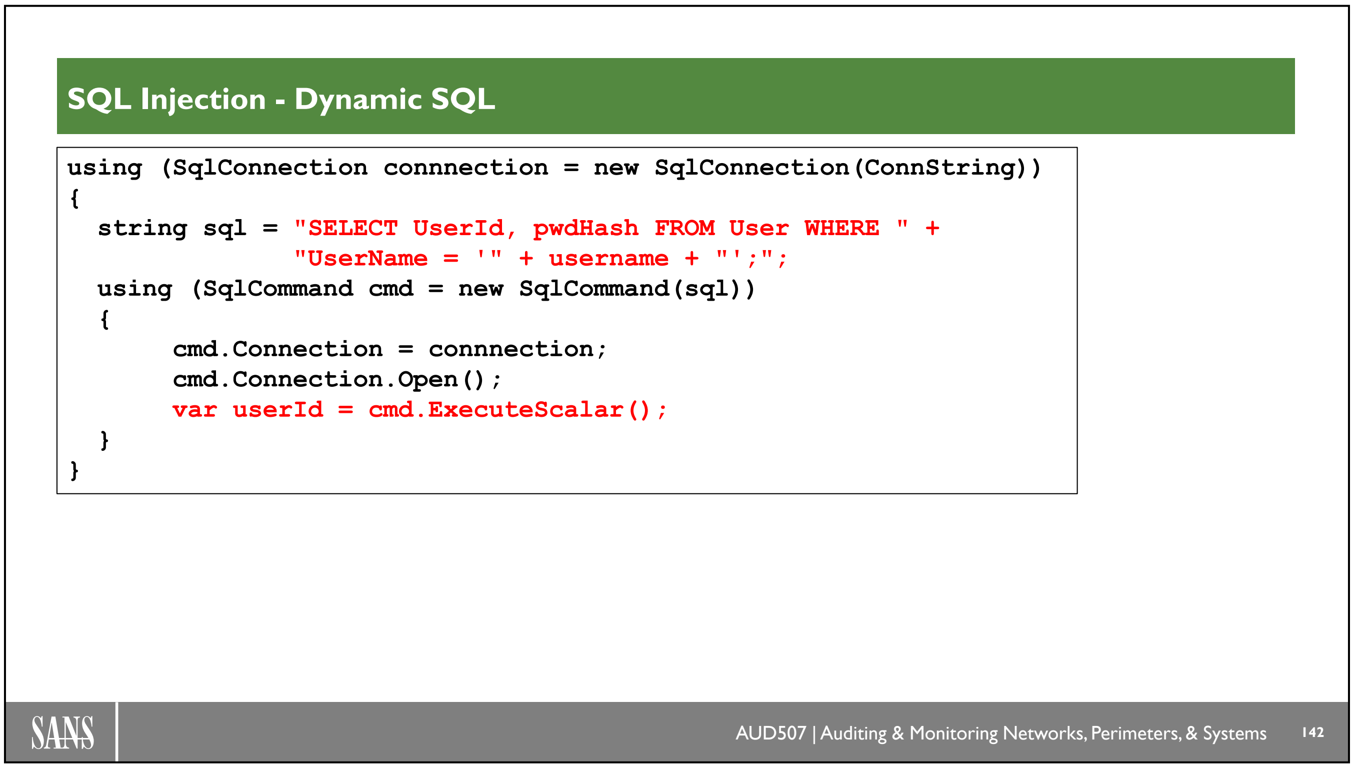
The web developer has created a simple HTML form which uses the POST HTTP method to send a username and password to the backend of the application for processing. Simplified source code for the server-side code is on the next page.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



SQL Injection - Dynamic SQL

```
using (SqlConnection connnection = new SqlConnection(ConnString))
{
  string sql = "SELECT UserId, pwdHash FROM User WHERE " +
               "UserName = '" + username + "';";
  using (SqlCommand cmd = new SqlCommand(sql))
  {
       cmd.Connection = connnection;
       cmd.Connection.Open();
       var userId = cmd.ExecuteScalar();
  }
}
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 142

This page has the simplified C# .NET code for the login function of our web application. The **username** variable contains the data passed from the web browser from the textbox in the web form.

The developer uses the user input as part of the SQL SELECT query used to authenticate the user. The code builds a string containing the entire SQL statement, including the user input, and then passes that string off to the database to be executed.

This programming technique is known as "dynamic SQL," because the ultimate content of the SQL statement is not known until runtime when the code processes the user input. Think of it this way: in dynamic SQL, the developer writes part of the database query, and the end user writes the rest of it…

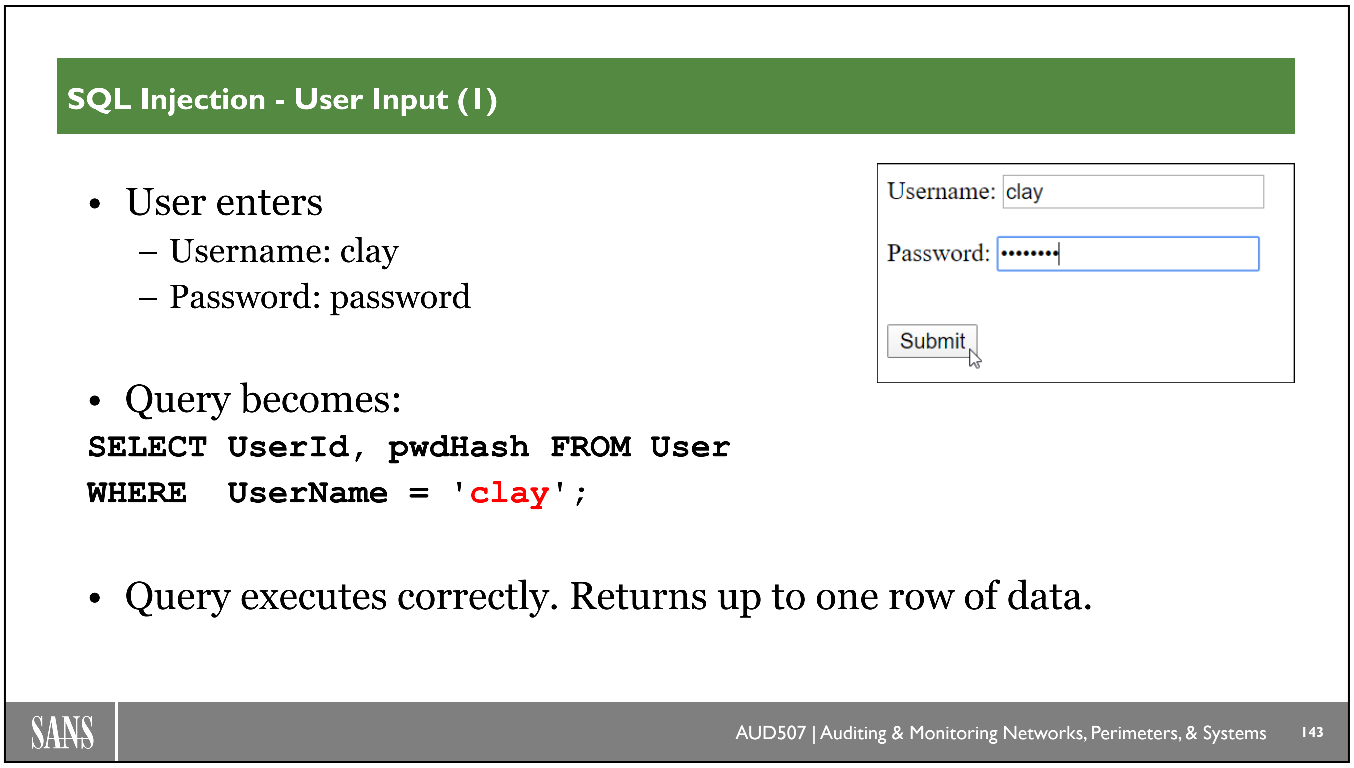
Sample code in this section is based on examples used by James Jardine in the SANS Software Security Blog.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

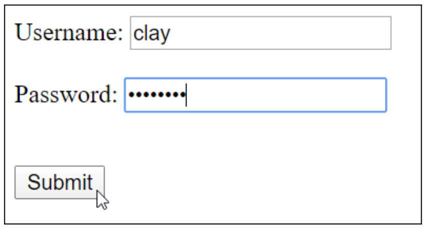
August 10, 2021

© SANS Institute 2021



SQL Injection - User Input (1)

- User enters
  - Username: clay
  - Password: password



Username: clay

Password: ••••••••

- Query becomes:

```
SELECT UserId, pwdHash FROM User
WHERE  UserName = 'clay';
```

- Query executes correctly. Returns up to one row of data.

When a user enters expected values for the username, the dynamic SQL statement contains a valid query, and the database returns proper results. In this case, the database would return a valid UserID if the username is correct, and would return NULL, or no value, if it is incorrect.

This is expected behavior and the application functions normally with this input.

August 10, 2021

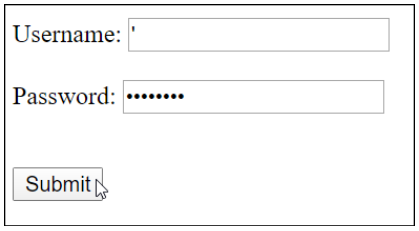
August 10, 2021

Technet24

© SANS Institute 2021

SQL Injection - User Input (2)

- User enters
  - Username: '
  - Password: password



- Query becomes:

```
SELECT UserId, pwdHash FROM User WHERE UserName = ''';
```

- Unterminated string constant SQL error! Why?

If the user enters UNEXPECTED input into the textbox, things can start to go wrong with our application. Because the user entered a single quotation mark for their username, our dynamic SQL query now contains a syntax error. We have an imbalance in quotes, resulting in the database throwing an "unterminated string constant," or similar, error.

If the application displays the error to the attacker (which we learned not to do earlier in our material), it will be easy for the attacker to determine that there is an injection vulnerability. Even if the application properly obscures the error, the attacker might notice that the results of the query have changed from those given for valid input.

The attacker's next step might be to try to exploit the injection flaw to return more data than expected by the developer.

August 10, 2021

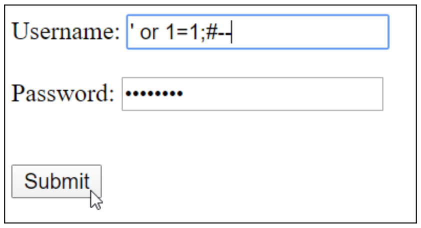
© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

SQL Injection - User Input (3)

- User enters
  - Username: ' or 1 = 1;#--
  - Password: password



- Query becomes:

```
SELECT UserId, pwdHash FROM User WHERE  UserName = ''
or 1 = 1;#--~~';~~
```

- May run correctly and return all rows! Why?

One technique used by attackers to return extra results from a dynamic query is to inject code that will always evaluate to TRUE. This may cause the database server, as it examines rows in the table to see if they match the query, to decide that ALL rows match and should be returned.

A common equality equation used in SQL injection is **or 1 = 1**. This attack leverages basic Boolean logic to create a WHERE clause that is always true. There is a problem for the attacker in this example though. Because of the single quotes used in the query, the attacker may need to get creative to ensure that there are no more invalid string errors. The characters at the end of the username string see this:

The semicolon is used to terminate one SQL statement and, optionally, begin another.

The # and **--** characters are used to create comments in commonly used SQL dialects, rendering the rest of the query as non-executable.

August 10, 2021

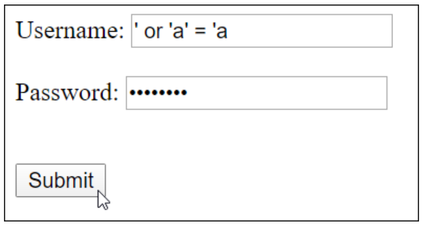
August 10, 2021

Technet24

© SANS Institute 2021

SQL Injection - User Input (4)

- User enters
  - Username: ' or 'a' = 'a
  - Password: password



- Query becomes:

```
SELECT UserId, pwdHash FROM User WHERE  UserName = '' or
'a' = 'a';
```

- Returns all rows with no errors! Why?

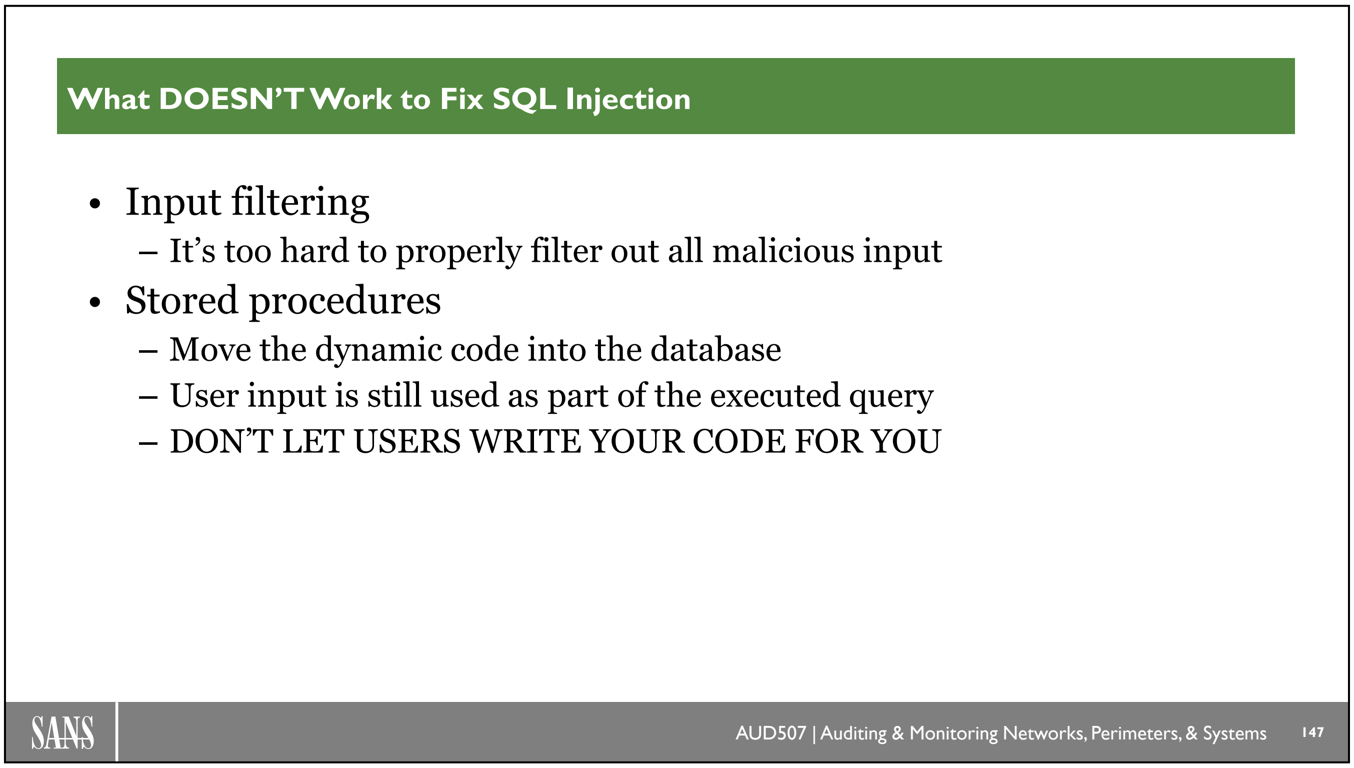
In this instance, the attacker has avoided having to use the semicolon as a statement terminator AND the # and **--** comment characters by simply using the last single quotation mark as part of their design. By omitting a single-quote from the injected code, they successfully consume the last single quote from the programmer's code.

This injection will work even on SQL servers which have been configured not to allow multiple queries (separated by a semicolon) on a single line. It should return values from all rows in the **User** table, regardless of the username and password stored in each row.

August 10, 2021

August 10, 2021

© SANS Institute 2021



What DOESN'T Work to Fix SQL Injection

- Input filtering
  - It's too hard to properly filter out all malicious input
- Stored procedures
  - Move the dynamic code into the database
  - User input is still used as part of the executed query
  - DON'T LET USERS WRITE YOUR CODE FOR YOU

There is a LOT of bad advice available on the internet about how to correct SQL injection flaws. It saddens me to say it, but even Little Bobby Tables' mother incorrectly recommends input sanitization as a solution in the famous XKCD 'Exploits of a Mom' comic.

Input filtering—both accept-listing and reject-listing—can and often will be defeated by determined attackers. It's just too difficult to anticipate and defend against every possible dangerous input string.

Moving your dynamic code into a stored procedure—a way of saving executable code in the database, rather than as part of the web application—is also doomed to fail. The problem lies not in where the code is stored, but in the fact that when you use dynamic SQL, you are allowing strangers on the internet (users and attackers) to write part of your database code.

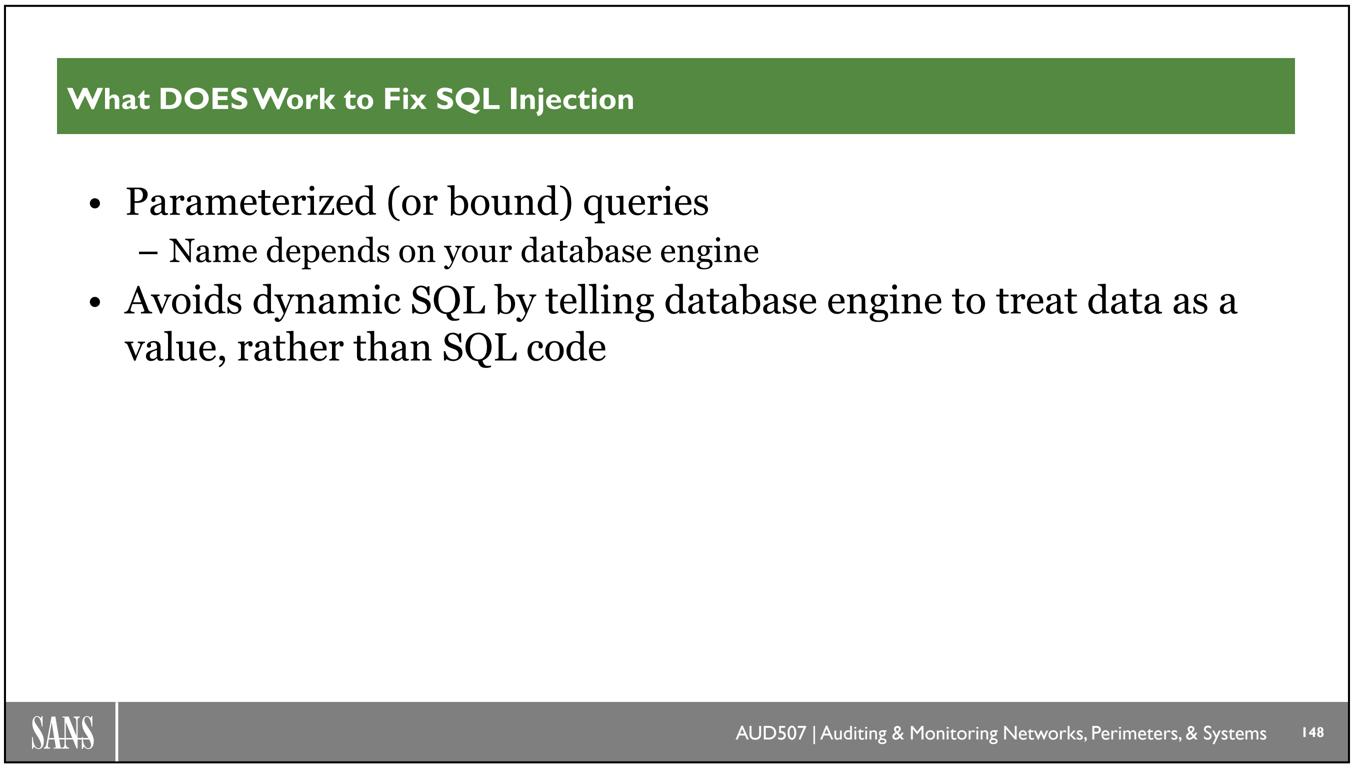
Parameterized, or bound (the name depends on your database vendor), queries, outlined on the next page, are the ONLY sure-fire way to prevent SQL injection flaws in your code.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



What DOES Work to Fix SQL Injection

- Parameterized (or bound) queries
  - Name depends on your database engine
- Avoids dynamic SQL by telling database engine to treat data as a value, rather than SQL code

SANS

AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 148

In a parameterized query, the developer specifies that part of the query (a parameter) is reserved to contain data, rather than executable code. The parameters are treated as literal values when processed by the query engine. If the user enters **‘ or 1=1;--** in the username field, and this value is passed in a parameter, the database will search for rows where the username column contains a LITERAL value of **‘ or 1=1;--**. Because the user data is not treated as executable, the injection flaw is completely fixed.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021

Parameterized Query Example

```
using (SqlConnection connection = new SqlConnection(ConnString))
{
  // Build the query statement using user input.
  string sql = "SELECT UserId, pwdHash FROM User WHERE " +
               "UserName = @username";
  using (SqlCommand cmd = new SqlCommand(sql))
  {
       cmd.Parameters.AddWithValue( "@username", username );
       var userId = cmd.ExecuteScalar();
  }
}
```

In the sample C# .NET code above, **@username** and **@password** are the names of the parameters being used in the SQL query. The user input passed into the program as the **username** and **password** variables are copied into the parameters using the **Parameters.AddWithValue** function of the SQLCommand object.

Remember that these parameters can only contain data and will never be treated as code by the database, thus rendering the user input safe to use.

August 10, 2021

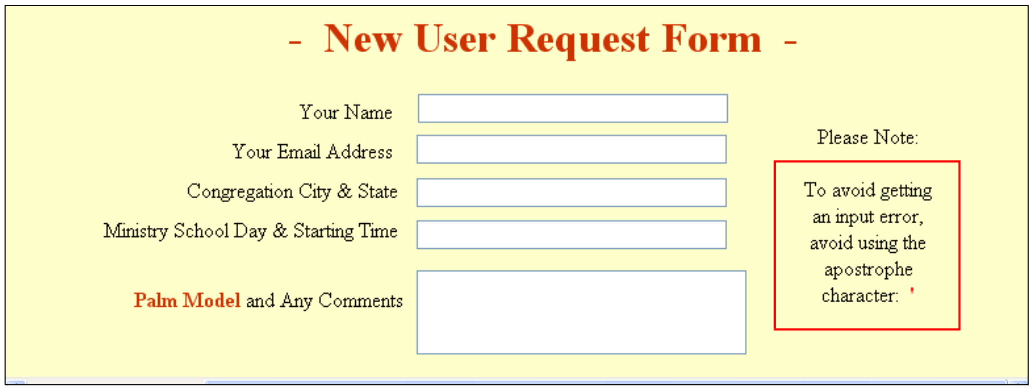
August 10, 2021

Technet24

© SANS Institute 2021

What's Worse?

- Having a flaw and not knowing
- Knowing your app is broken and not fixing it



- New User Request Form -

Your Name	
Your Email Address	
Congregation City & State	
Ministry School Day & Starting Time	
Palm Model and Any Comments	

Please Note:

To avoid getting an input error, avoid using the apostrophe character: '

Here's a good question to consider, though. What's worse, having a web application with a critical flaw and being unaware of it, or knowing that you have a critical flaw and asking people not to exploit it? Here's a tip: People won't listen to you. If you have a flaw, you're in for trouble, it's just a matter of when.

Look at this real webpage from the internet where the programmer recommends that you not send single quotes so that you can avoid generating an error message. Let's use this to demonstrate some other problems.

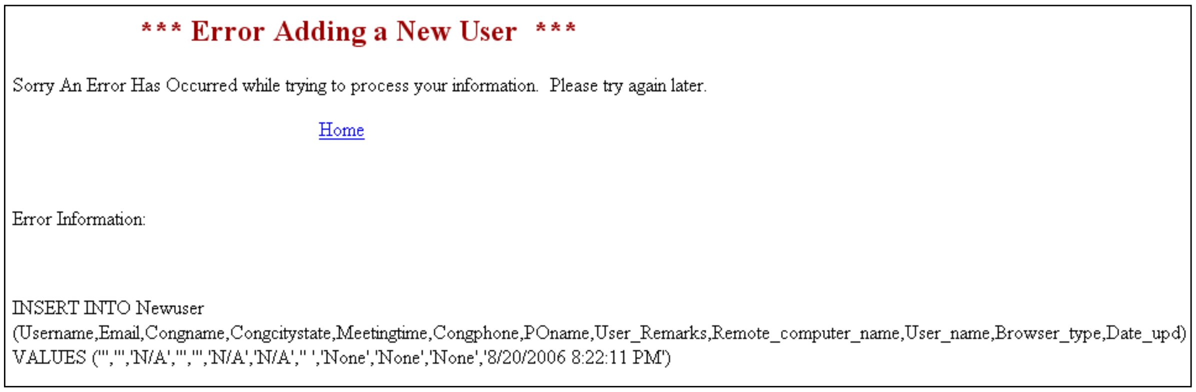
August 10, 2021

August 10, 2021

© SANS Institute 2021

Oops!

- How's this for an error



*** Error Adding a New User ***

Sorry An Error Has Occurred while trying to process your information. Please try again later.

Home

Error Information:

```
INSERT INTO Newuser
(Username,Email,Congname,Congcitystate,Meetingtime,Congphone,POname,User_Remarks,Remote_computer_name,User_name,Browser_type,Date_upd)
VALUES ("","",'N/A',"","",'N/A','N/A',"  ",'None','None','None','8/20/2006 8:22:11 PM')
```

In this particular case, if you do send a single quote (in this case we sent a single quote in every field), we got a very handy error message back. Not only do we know that we had a database error, but the code was helpful enough to show us *exactly* how what we sent in was used! At the bottom of the page, you can clearly see the actual SQL that the server tried to execute!

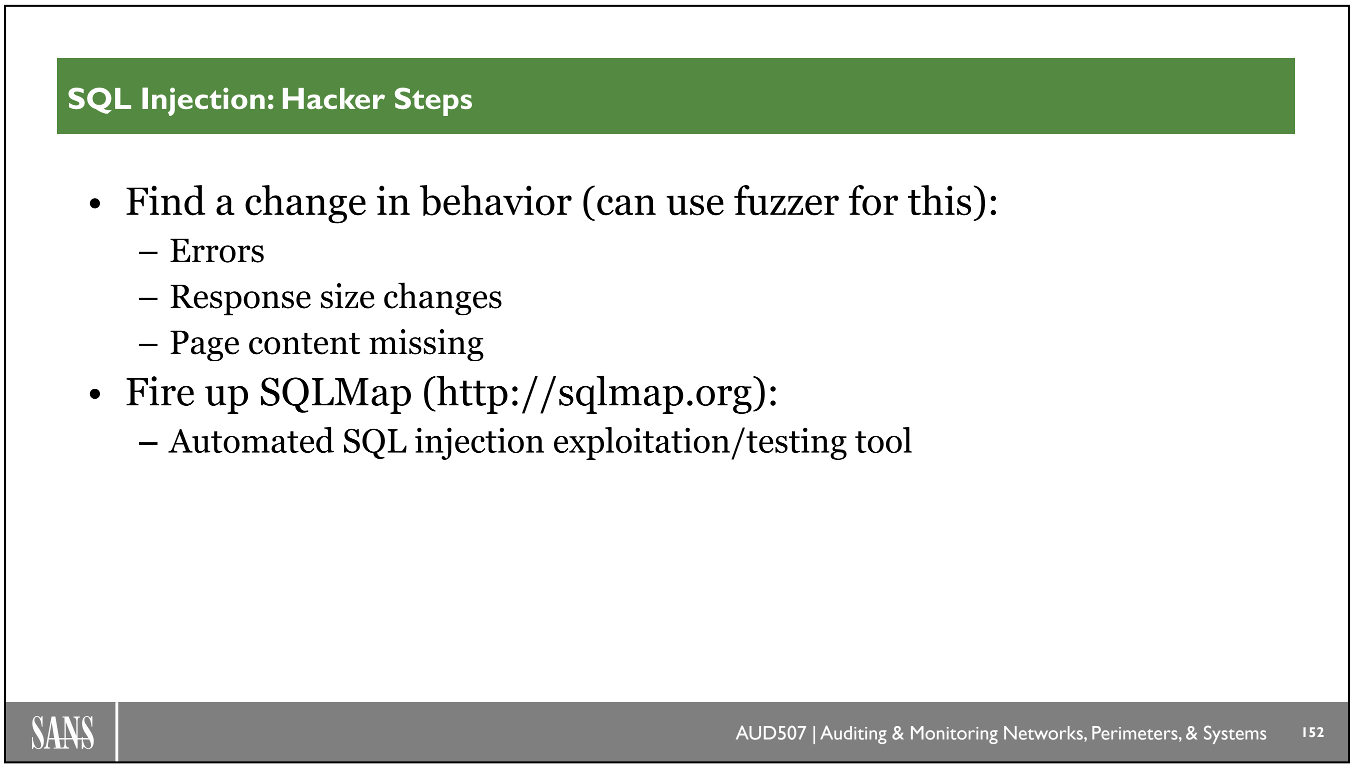
This is clearly bad. Now the attacker can easily craft extremely malicious requests to download copies of data, cause the server to initiate outbound connections to other sites to send off copies of its data, drop tables, drop databases, and more. You name it. We can use this to illustrate one other serious issue and our last topic for this section.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



SQL Injection: Hacker Steps

- Find a change in behavior (can use fuzzer for this):
  - Errors
  - Response size changes
  - Page content missing
- Fire up SQLMap (http://sqlmap.org):
  - Automated SQL injection exploitation/testing tool

In this class, we have no need to proceed any further than identifying the change in behavior or possibly generating the SQL error. What, though, would an attacker do with this information?

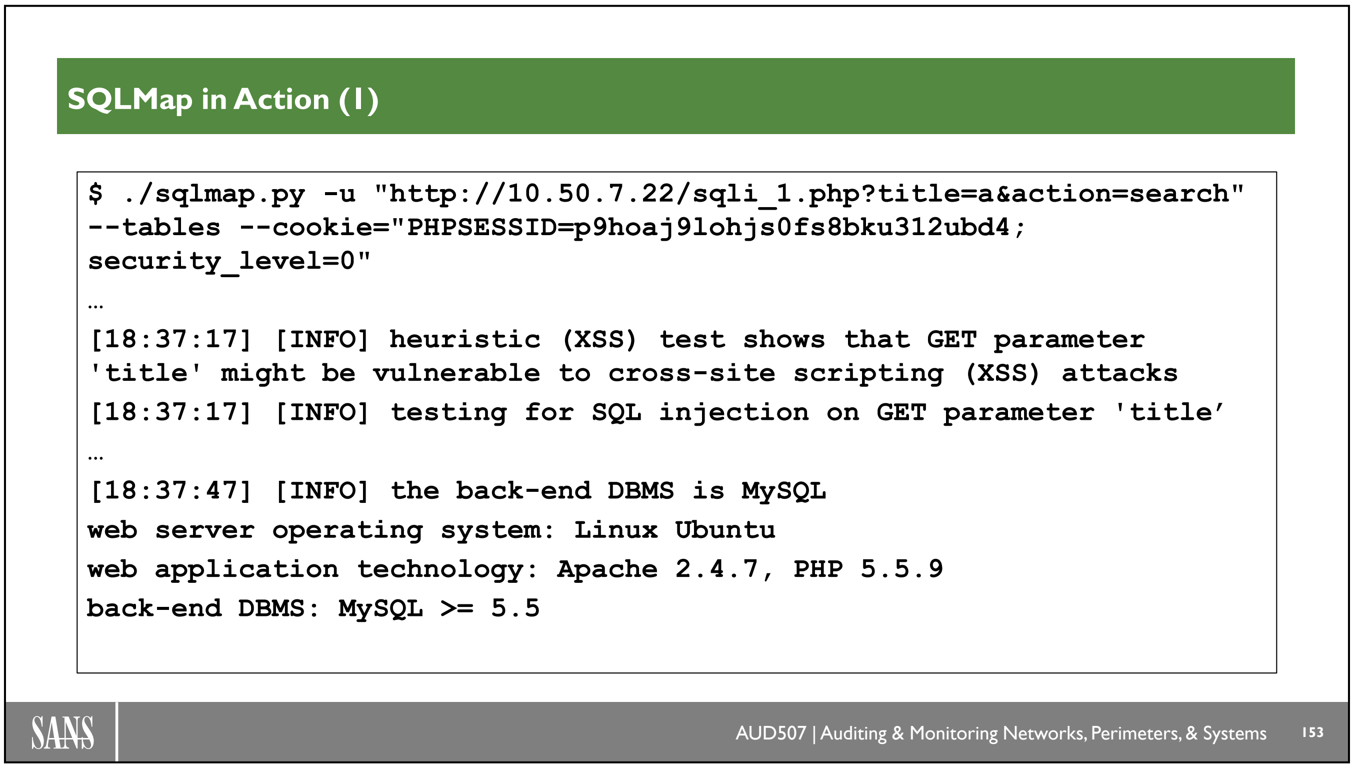
To ease exploitation an attacker might use a tool such as SQLMap to further test and subsequently exploit a vulnerable application. Using SQLMap the attacker can take the query that we have identified as causing an unexpected behavior or possibly a SQL error and automatically test using a wide variety of SQL injection techniques. These range from basic injection testing to advanced timing-based attacks that allow the tool to blindly retrieve data using boolean testing of the application responses.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



SQLMap in Action (1)

```
$ ./sqlmap.py -u "http://10.50.7.22/sqli_1.php?title=a&action=search"
--tables --cookie="PHPSESSID=p9hoaj9lohjs0fs8bku312ubd4;
security_level=0"
…
[18:37:17] [INFO] heuristic (XSS) test shows that GET parameter
'title' might be vulnerable to cross-site scripting (XSS) attacks
[18:37:17] [INFO] testing for SQL injection on GET parameter 'title'
…
[18:37:47] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.7, PHP 5.5.9
back-end DBMS: MySQL >= 5.5
```

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 153

In this screenshot, SQLMap is being used to test the GET parameters of a web application for SQL injection vulnerabilities. We omitted a LOT of the output for brevity, but you can see in the included output, that SQLMap identified the "title" parameter as being vulnerable to injection, and then proceeded to send multiple (in this case, over 700) subsequent requests to enumerate the database schema and details of the database server being used with the web application.

In this portion of the output SQLMap has identified that the target is a MySQL (MariaDB) server running on Ubuntu Linux. It has also identified the Apache version (2.4.7) and PHP version (5.5.9) in use by the web application.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

SQLMap in Action (2)

```
sqlmap identified the following injection point(s) with a total of 763
HTTP(s) requests:
…
[18:37:47] [INFO] fetching database names
[18:37:47] [INFO] fetching tables for databases: 'bWAPP,
information_schema, mysql, performance_schema'
Database: mysql
[24 tables]
+-----------------------------------------------+
| db                                            |
| event                                         |
| user                                          |
…
```

In this portion of the output SQLMap has begun identifying the schema of the database in use. It is giving the user a list of all the tables in the current database, which is named “bWapp.”

It is possible with SQLMap to dump all the data in all the tables (using the --dump command-line option), and even to attempt to crack any hashed passwords saved in the database. We strongly recommend that you avoid dumping any sensitive data from any databases during application testing. There is usually a safer way to prove that it is possible to obtain access without losing your job over a breach of confidential data!

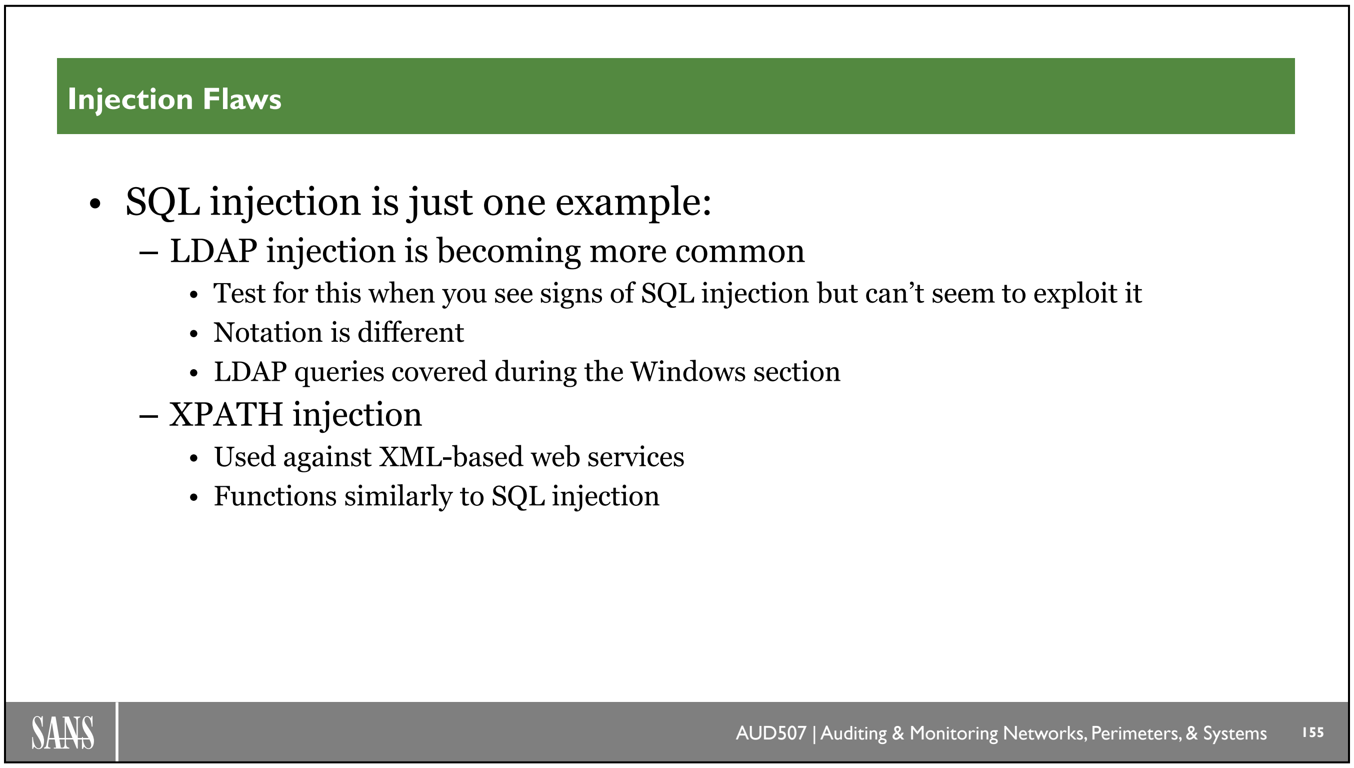
Total elapsed time for this test was under 30 seconds. Some of that time was spent waiting for user input to complete the testing steps.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Injection Flaws

- SQL injection is just one example:
  - LDAP injection is becoming more common
    - Test for this when you see signs of SQL injection but can't seem to exploit it
    - Notation is different
    - LDAP queries covered during the Windows section
  - XPATH injection
    - Used against XML-based web services
    - Functions similarly to SQL injection

This covers the basics of how SQL injection occurs. This is not, however, the only kind of injection flaw that might exist!

LDAP injection is becoming more and more common. We're not going to dig deeply into LDAP injection here. Instead, we have a more complete discussion of LDAP and LDAP query notation during the Windows section of the course, to which we can apply these same principles.

XPATH injection is frequently used against web services which transmit data via XML documents. XPATH can be used to "query" documents to retrieve specified data and can be subject to injection attacks much like SQL, often resulting in data leakage from the application.

How do I know which kind of injection flaw I might have? First, it's not critical that an auditor can distinguish the two. However, if you can tell one from the other, it can have an enormous impact on the acceptance of your findings! The simplest advice that we can give in differentiating the two is that if you find a change in behavior when trying SQL injection techniques but are unable to exploit it, consider adding LDAP style qualifiers and see if those are more successful.

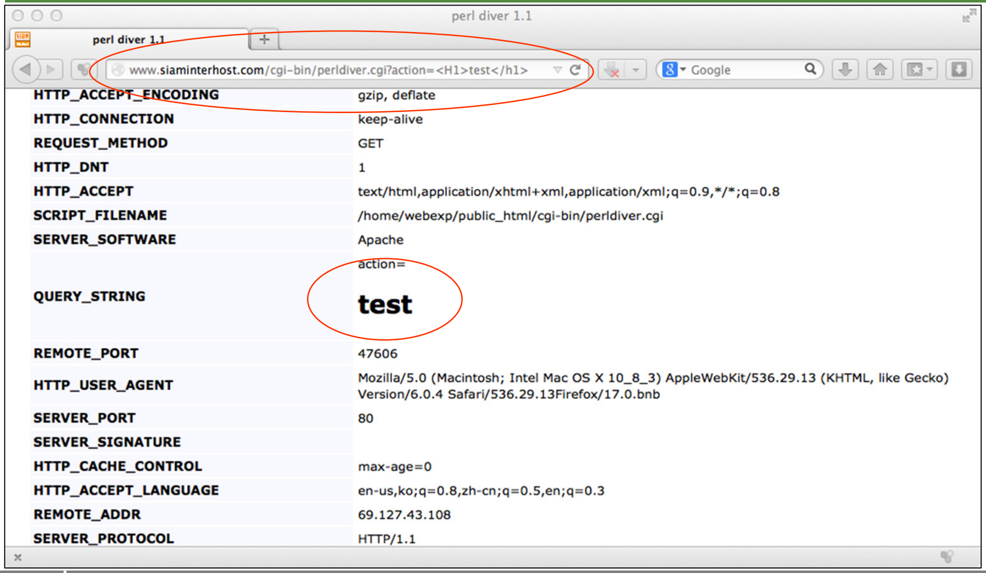
August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Injection Flaws: HTML Injection


perl diver 1.1
perl diver 1.1
www.siaminterhost.com/cgi-bin/perldiver.cgi?action=<H1>test</h1>
Google

HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_CONNECTION	keep-alive
REQUEST_METHOD	GET
HTTP_DNT	1
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
SCRIPT_FILENAME	/home/webexp/public_html/cgi-bin/perldiver.cgi
SERVER_SOFTWARE	Apache
QUERY_STRING	action= **test**
REMOTE_PORT	47606
HTTP_USER_AGENT	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.29.13 (KHTML, like Gecko) Version/6.0.4 Safari/536.29.13Firefox/17.0.bnb
SERVER_PORT	80
SERVER_SIGNATURE	
HTTP_CACHE_CONTROL	max-age=0
HTTP_ACCEPT_LANGUAGE	en-us,ko;q=0.8,zh-cn;q=0.5,en;q=0.3
REMOTE_ADDR	69.127.43.108
SERVER_PROTOCOL	HTTP/1.1

Injection flaws, in general, are all about convincing something (like a web browser) to interpret what should have been data in some other context. In the slide, for instance, we can see that we have substituted HTML into the query argument. What's the result? Can you see what this has caused to happen to the word "test"? The content <H1>test</h1> was submitted as data, but because of how the application sends back the content to the browser for rendering, it is interpreted as *code* rather than data.

What if we were to send something more significant than some innocuous HTML?

August 10, 2021

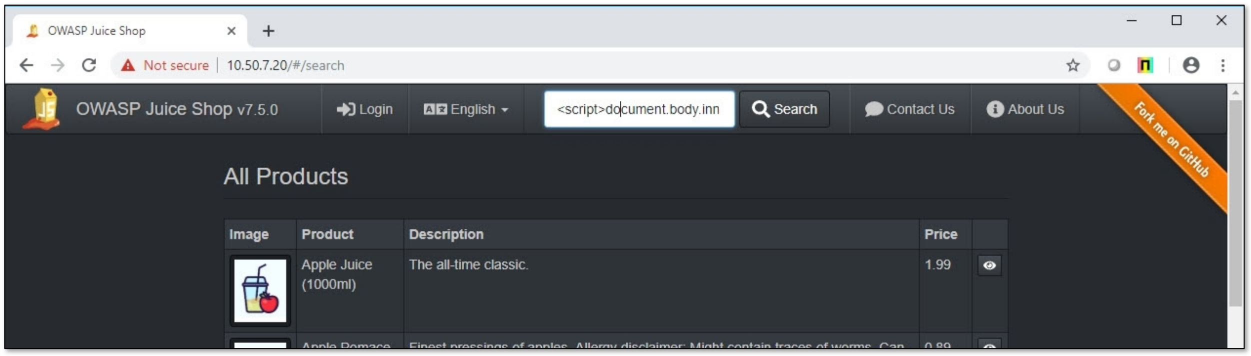
August 10, 2021

© SANS Institute 2021

Cross-Site Scripting: XSS

- HTML Injection + JavaScript = XSS

```
<script>document.body.innerHTML=document.cookie;</script>
```



All Products

Image	Product	Description	Price
	Apple Juice (1000ml)	The all-time classic.	1.99
	Apple Pomace	Finest pressings of apples. Allergy disclaimer: Might contain traces of worms. Can	0.89

Just reformatting text on a page is mildly interesting, but what happens when we follow the example of SQL injection and insert actual code? Specifically, what could we do if we injected JavaScript? The easy answer is, "Bad things." XSS is, unfortunately, common to find. For our example, we'll use the Juice Shop application from our lab VMs.

For XSS to be successful, the client side must send code to the server that will eventually be reflected back to the client. That reflection might occur as a result of clicking on a URL, opening an HTML email, or looking at a posting on a web blog. Whatever the case, someone has found a site where the page will store and return exactly what is sent in with no filtering.

There is one other requirement. There must actually be something useful on that page that the attacker wants. This could be a session ID (in a URL, hidden form element, or cookie) or some piece of sensitive information such as a credit card number. These can, of course, be stored in the same ways.

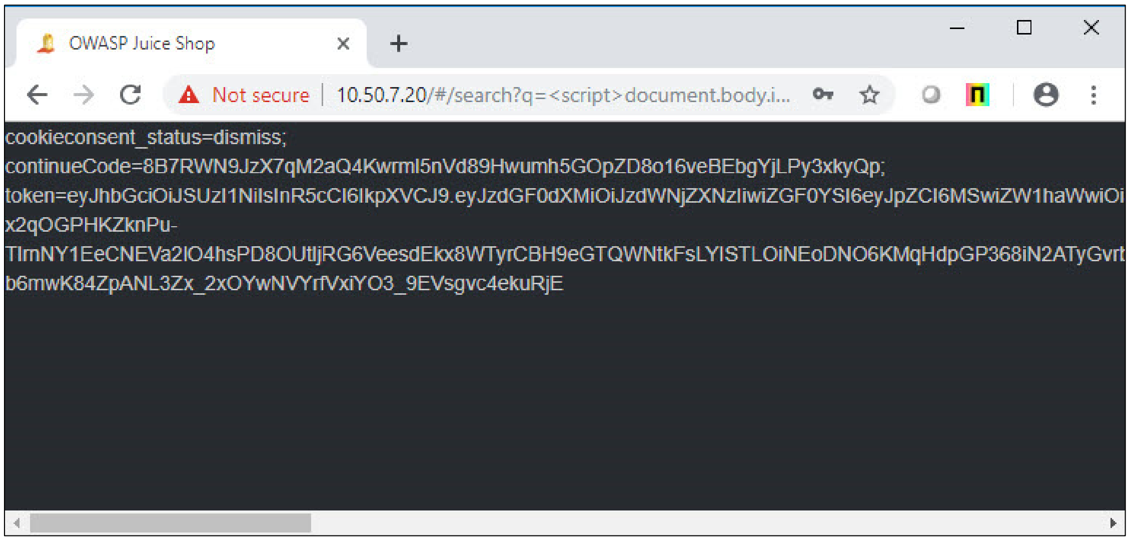
August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

Vulnerable!



```
cookieconsent_status=dismiss;
continueCode=8B7RWN9JzX7qM2aQ4Kwrml5nVd89Hwumh5GOpZD8o16veBEbgYjLPy3xkyQp;
token=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwiZW1haWwiOi
x2qOGPHKZknPu-
TImNY1EeCNEVa2lO4hsPD8OUtljRG6VeesdEkx8WTyrCBH9eGTQWNtkFsLYISTLOiNEoDNO6KMqHdpGP368iN2ATyGvrt
b6mwK84ZpANL3Zx_2xOYwNVYrfVxiYO3_9EVsgvc4ekuRjE
```

Here you see the results of our injected JavaScript. All of the cookies for the page are displayed on the screen. This happens because the test we searched for in the search box was:

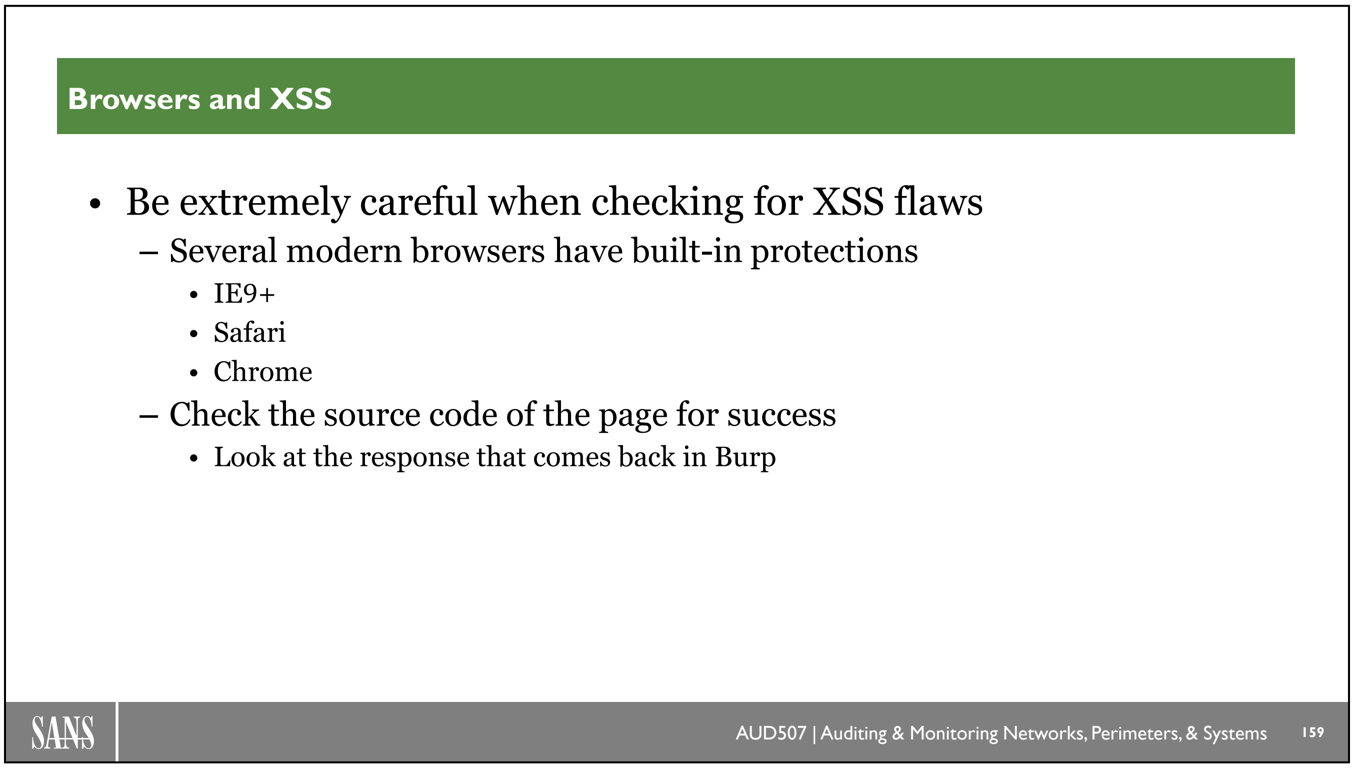
```
<script>document.body.innerHTML=document.cookie;</script>
```

In this case, we replaced the entire content of the webpage with the cookies associated with the current user. What if we had sent those cookies off to an attacker? They could now interact with this website using the victim's valid session – effectively impersonating the victim on the application.

August 10, 2021

August 10, 2021

© SANS Institute 2021



Browsers and XSS

- Be extremely careful when checking for XSS flaws
  - Several modern browsers have built-in protections
    - IE9+
    - Safari
    - Chrome
  - Check the source code of the page for success
    - Look at the response that comes back in Burp

It is extremely important to note that recent changes to browsers have served to increase security. This same improvement, however, can cause us to overlook XSS errors in our sites!

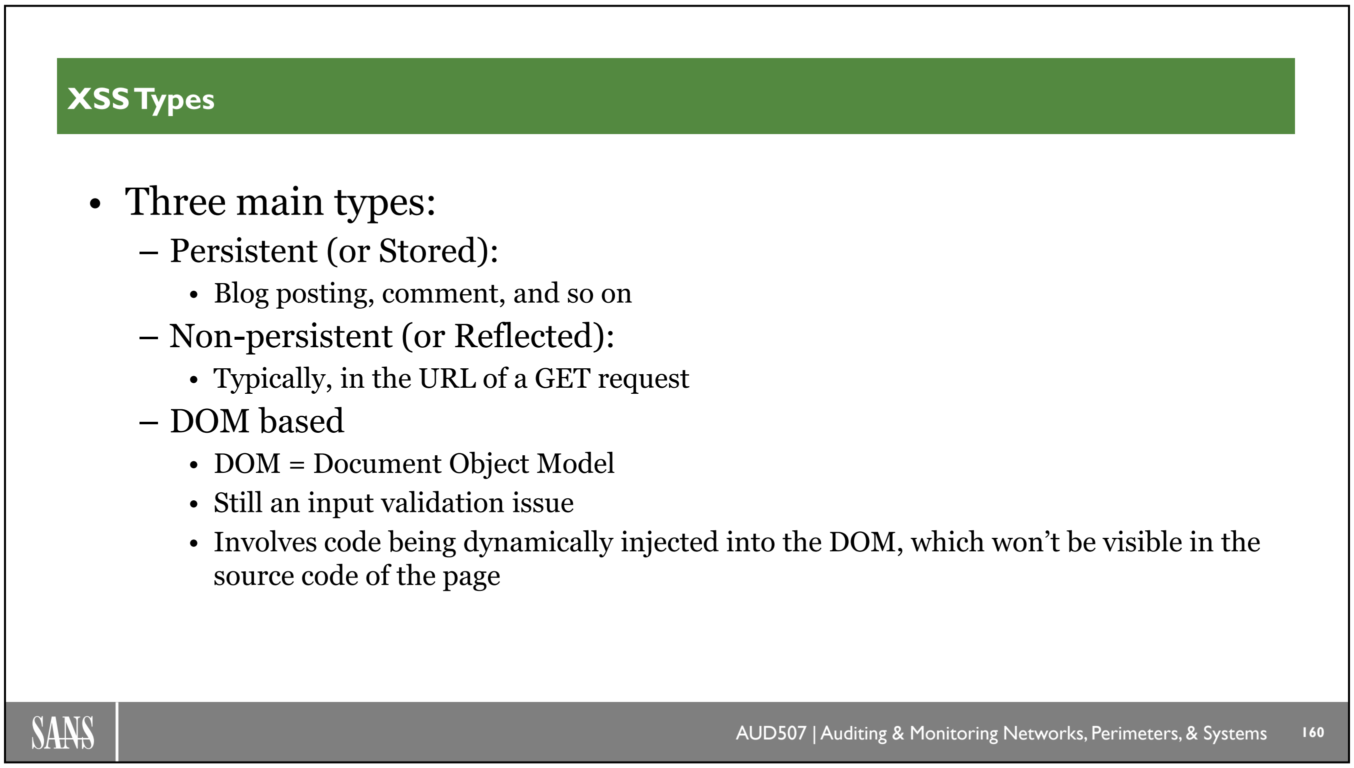
The newest browsers have basic XSS protections built in. Although this is fantastic news, we have no guarantees when it comes to the version or brand of browser that our users might be using. If we do all our testing with the most modern version of the browser, our site might actually have a flaw that we will not see because of our new browser! As a result, we suggest that when you test for XSS problems you should actually view the source code for the page to see if the JavaScript was sent to the browser rather than relying on a window opening.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



XSS Types

- Three main types:
  - Persistent (or Stored):
    - Blog posting, comment, and so on
  - Non-persistent (or Reflected):
    - Typically, in the URL of a GET request
  - DOM based
    - DOM = Document Object Model
    - Still an input validation issue
    - Involves code being dynamically injected into the DOM, which won't be visible in the source code of the page

There are three main types of XSS flaws today. The first type is a Persistent or Stored XSS flaw. This means that the application accepts input that will be stored, likely in a database, and later displayed back to users. You could imagine this being a situation in which a blog enables a user to submit unfiltered input that will later be displayed back to other users who view that blog article. When the other users view the article, the code that the first user submitted will be sent to them and executed within their browser.

The second type is Non-persistent or Reflected. In this case, the attack vector is usually through a GET request, though this is not a requirement. Imagine a site that accepts a search term and that the search term is then "reflected" back to the user and displayed on the results page. If that search term is sent back without any filtering, it could easily lead to code execution within the context of the user's browser.

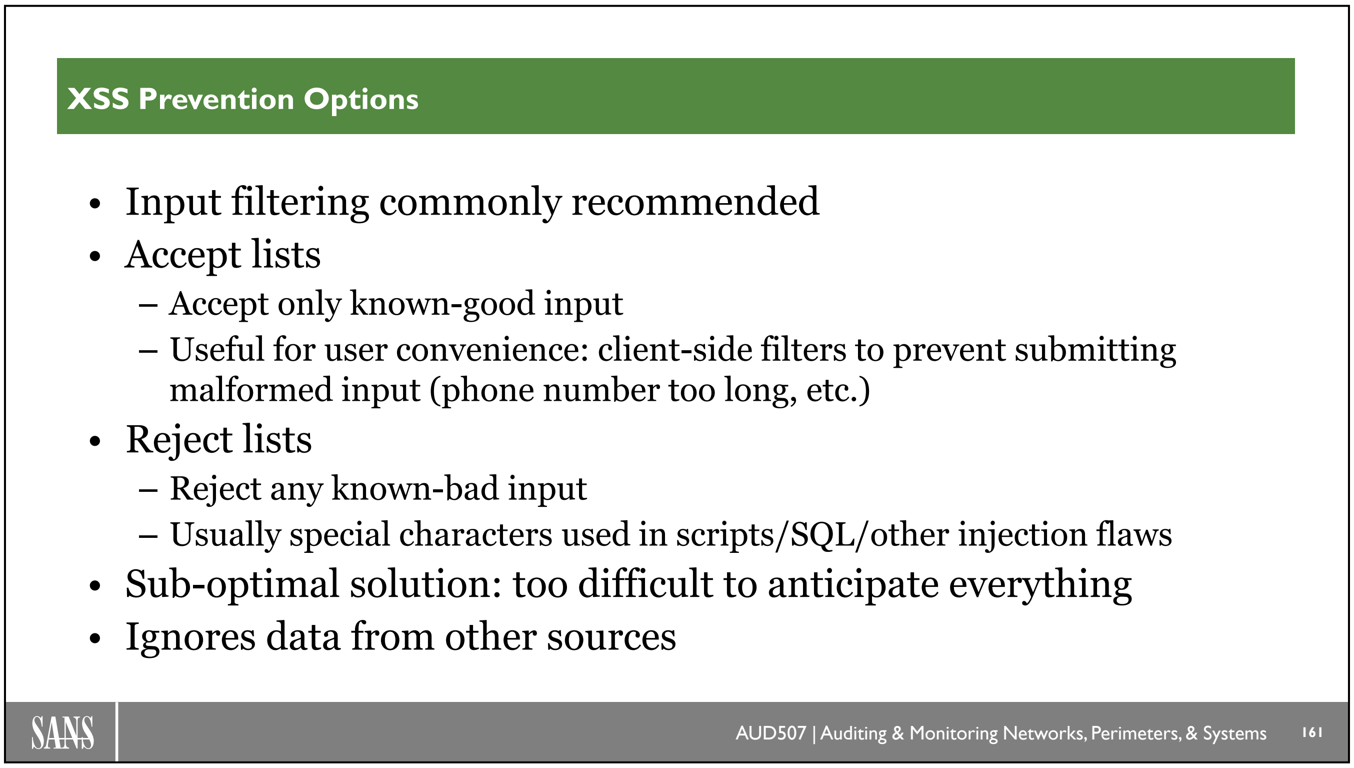
In DOM-based XSS, the script code doesn't even show up in the server response or the page's source code. The injection happens inside the JavaScript DOM. The script is still run, but it is much harder to detect. The first part of the next exercise will use a DOM-based attack.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



XSS Prevention Options

- Input filtering commonly recommended
- Accept lists
  - Accept only known-good input
  - Useful for user convenience: client-side filters to prevent submitting malformed input (phone number too long, etc.)
- Reject lists
  - Reject any known-bad input
  - Usually special characters used in scripts/SQL/other injection flaws
- Sub-optimal solution: too difficult to anticipate everything
- Ignores data from other sources

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 161

We routinely hear people recommend user input filtering as a way to prevent injection flaws. There are two problems with this advice: it ignores the fact that data can enter the web application from other sources than just user input, and it ignores processing which happens to data after receipt, which could render it dangerous after-the-fact!

While filtering WILL NOT prevent all attacks, you may still want to recommend it as an additional integrity control. Client-side code which warns about invalid data before it is submitted can make the data entry process more convenient for users. Filtering can be performed using "accept known good" or "reject known bad" techniques.

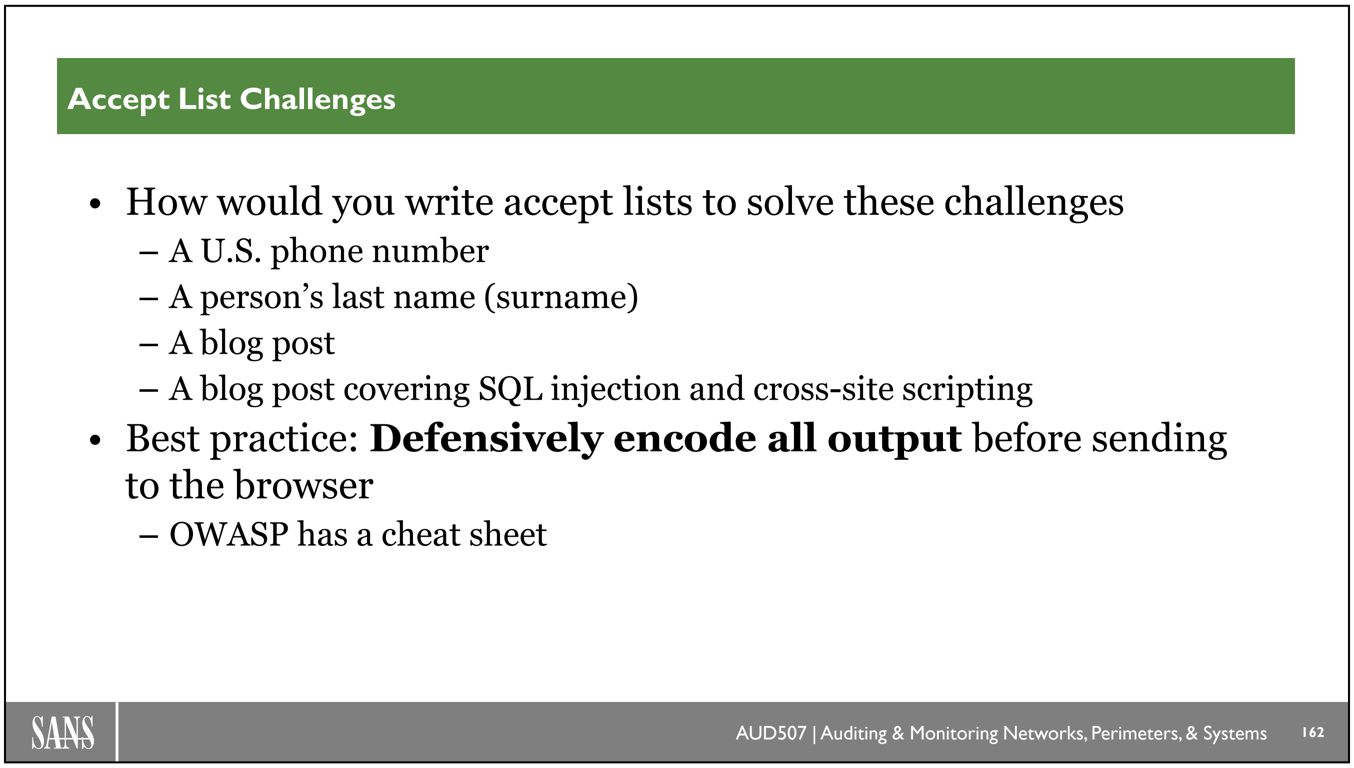
On the next slide, we demonstrate the difficulty of relying on filtering for complete protection, and then we tell you how to REALLY solve script injection flaws in your applications.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Accept List Challenges

- How would you write accept lists to solve these challenges
  - A U.S. phone number
  - A person's last name (surname)
  - A blog post
  - A blog post covering SQL injection and cross-site scripting
- Best practice: **Defensively encode all output** before sending to the browser
  - OWASP has a cheat sheet

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 162

Accept-listing is very useful for user convenience - preventing the user from entering input which will only be rejected later due to improper formatting. You can easily find yourself in a situation in which accept-listing is extremely difficult or possibly even impractical. By way of example, consider the questions on this slide. How would you design accept lists for these situations?

U.S phone numbers may not be that difficult. There will be 10 or 11 digits and some symbols like these: - + ( ) .

A person's surname is more difficult, given that you would need to think of valid characters for every language used for people's names.

The list for a blog post would include most punctuation in addition to alphanumeric characters, and if the blog post is about injection flaws, nearly all characters would need to be allowed.

The problem of accept-listing quickly becomes intractable. Instead, we should defend our applications when user data is **output** back to the client, by encoding the data using the appropriate scheme for the final use of the data. There are a large number of encoding schemes which can be used in URLs, HTML, HTTP headers, etc. OWASP maintains an excellent cheat sheet to inform your developers how to do encoding in most situations.

August 10, 2021

August 10, 2021

© SANS Institute 2021

Sensitive Output: Anticaching

- Map the web application by performing each transaction and view all pages with sensitive data:
  - Record all traffic (HTTP and HTML):
    - Using proxy like Burp
- For each instance of sensitive data displayed, determine if anti-caching techniques were used (next slide)

Here is yet another audit technique that requires us to walk the application and record all traffic for further analysis.

If any of these cached pages contain sensitive data, that might be a security issue (depending on how sensitive the data is).

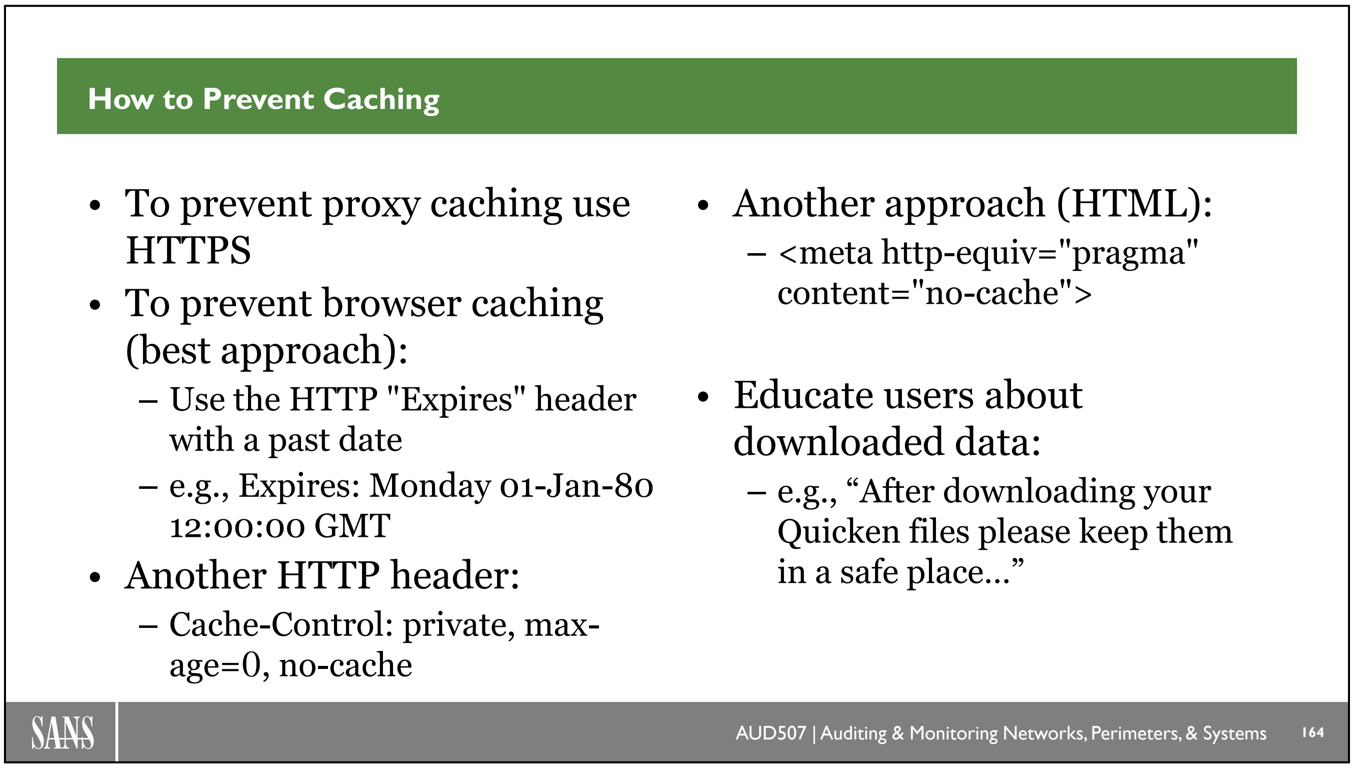
So, after you walk the site and manually mirror each page and transaction, how can you tell if the web server was trying to prevent your browser from caching? See the next slide for the answer.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



How to Prevent Caching

- To prevent proxy caching use HTTPS
- To prevent browser caching (best approach):
  - Use the HTTP "Expires" header with a past date
  - e.g., Expires: Monday 01-Jan-80 12:00:00 GMT
- Another HTTP header:
  - Cache-Control: private, max-age=0, no-cache
- Another approach (HTML):
  - <meta http-equiv="pragma" content="no-cache">
- Educate users about downloaded data:
  - e.g., "After downloading your Quicken files please keep them in a safe place..."

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 164

Rather than relying on the user to have the proper client-side settings, we focus our efforts on **server-side techniques** to try to prevent web browser caching.

The HTTP Expires header field gives the date and time after which the page should be considered stale. This allows information providers to suggest a date after which the information may no longer be valid. Web browsers prefer not to cache the page beyond the date given. Therefore, if the date given in the Expires header is equal to or earlier than the current date and time, the recipient (that is, web browser) will not cache the page.

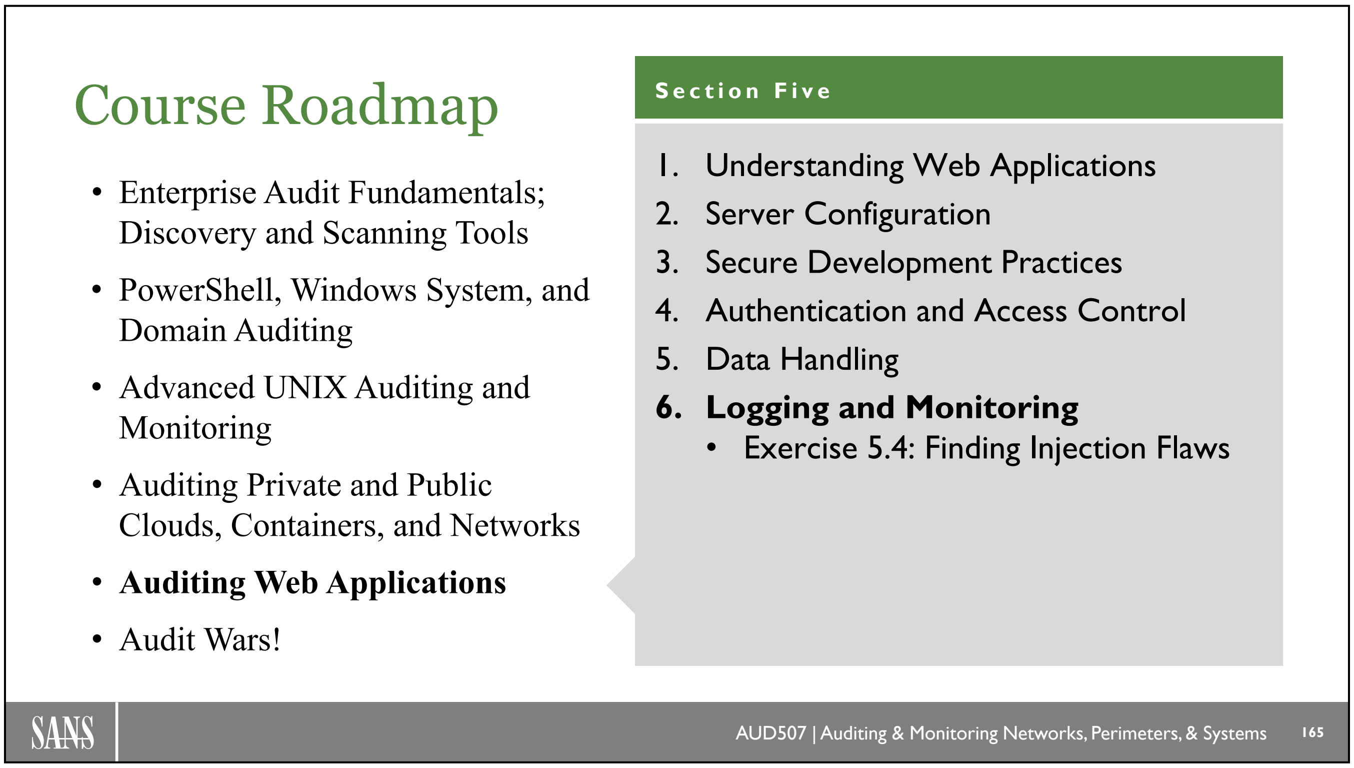
Another method that works well is to insert a <meta http-equiv='pragma' content='no-cache'> header into the HTML of the pages that you are sending to the client. Yet another method is to send a Cache-Control:private HTTP header using the server (or from the web application).

Alternatively, you can warn your users about clearing their browser's cache manually. Using server techniques to prevent caching is preferred. (Don't trust the user to do the right thing.)

August 10, 2021

August 10, 2021

© SANS Institute 2021



Course Roadmap

- Enterprise Audit Fundamentals; Discovery and Scanning Tools
- PowerShell, Windows System, and Domain Auditing
- Advanced UNIX Auditing and Monitoring
- Auditing Private and Public Clouds, Containers, and Networks
- **Auditing Web Applications**
- Audit Wars!

Section Five

1. Understanding Web Applications
2. Server Configuration
3. Secure Development Practices
4. Authentication and Access Control
5. Data Handling
6. **Logging and Monitoring**
    - Exercise 5.4: Finding Injection Flaws

SANS | AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 165

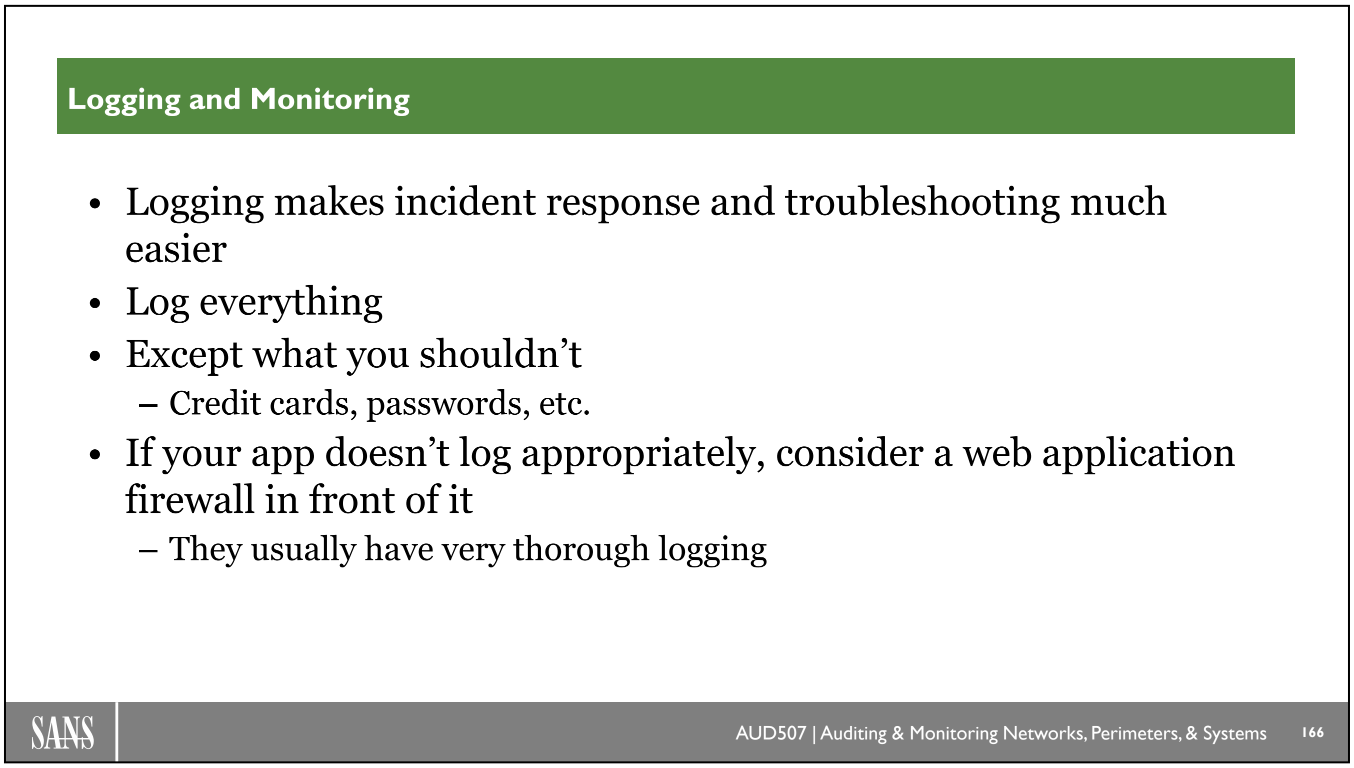
This page intentionally left blank.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Logging and Monitoring

- Logging makes incident response and troubleshooting much easier
- Log everything
- Except what you shouldn't
  - Credit cards, passwords, etc.
- If your app doesn't log appropriately, consider a web application firewall in front of it
  - They usually have very thorough logging

SANS AUD507 | Auditing & Monitoring Networks, Perimeters, & Systems 166

The last of our secure practices is logging and monitoring. To assist with troubleshooting problems and with handling security incidents, we recommend that you configure your application to log everything it can – with some notable exceptions. Sensitive data like passwords and credit card numbers should never be written to audit logs where it could be exposed to theft or abuse.

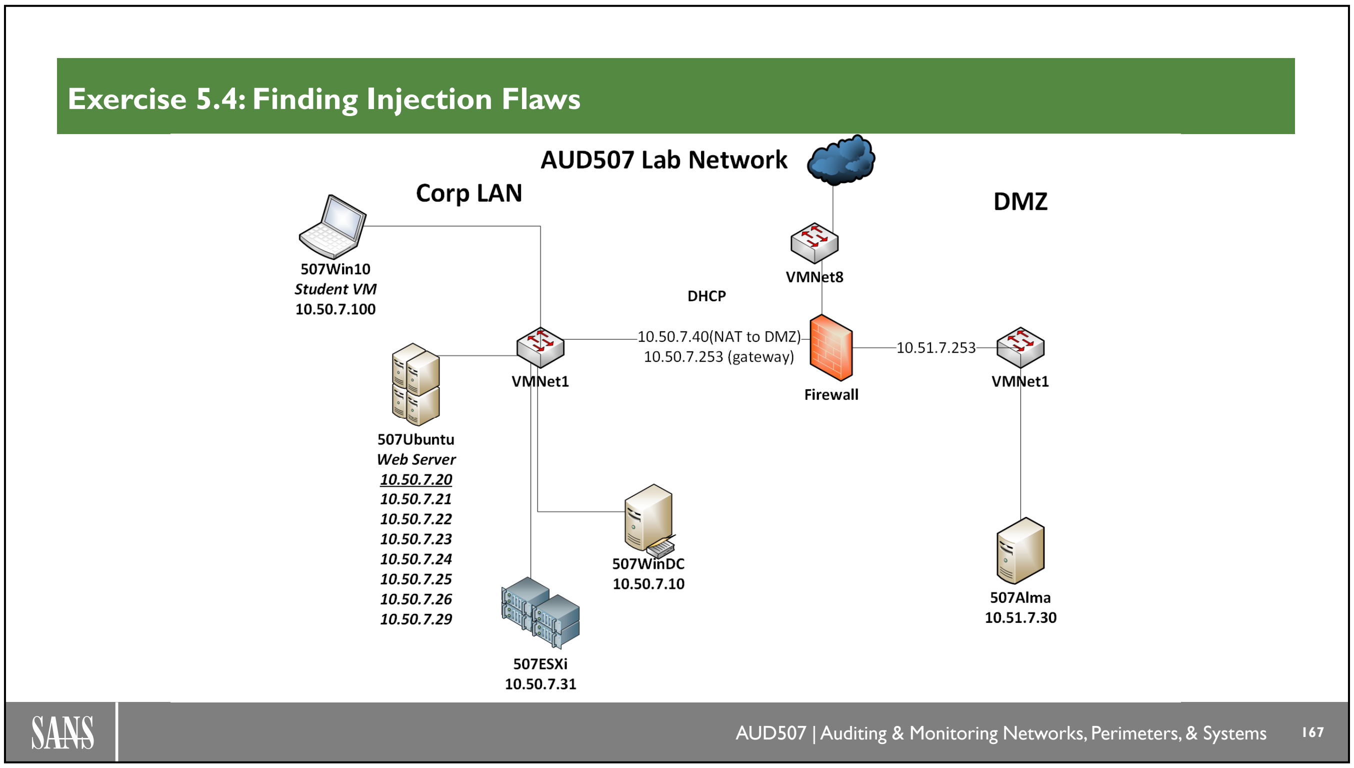
If your web application was written without a logging capability, add this function to later versions if possible. In the meantime, use the web application firewall in front of the application to handle your logging needs.

August 10, 2021

© 2021 Risenhoover Consulting, Inc.

August 10, 2021

© SANS Institute 2021



Exercise 5.4: Finding Injection Flaws

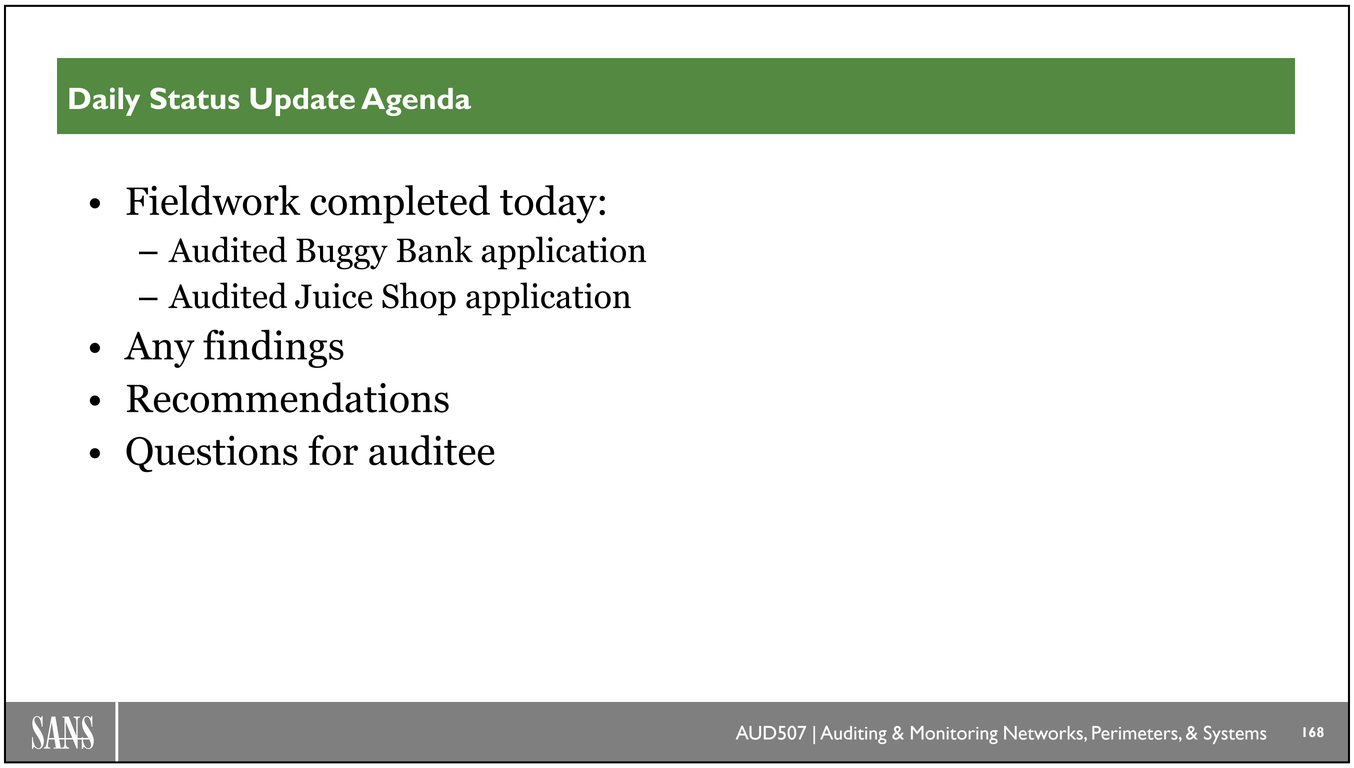
Please continue working in your workbook with Exercise 5.4: Finding Injection Flaws.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021



Daily Status Update Agenda

- Fieldwork completed today:
  - Audited Buggy Bank application
  - Audited Juice Shop application
- Any findings
- Recommendations
- Questions for auditee

Think back on the fieldwork you completed today.

What problems did you discover with the Buggy Bank application?

What problems did you notice with the Juice Shop application?

What recommendations would you make?

Are there any other questions you would ask the auditee?

August 10, 2021

August 10, 2021

© SANS Institute 2021

Thank You!

This brings us to the end of this section. If you are taking the class at a conference, please take a moment to complete an evaluation form. You will be given a different evaluation every day of the class.

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

This page intentionally left blank.

August 10, 2021

© SANS Institute 2021

Index

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021

© SANS Institute 2021

August 10, 2021

August 10, 2021

Technet24

© SANS Institute 2021

August 10, 2021

August 10, 2021