Modern Webapp Penetration Testing

Hands-on Doing.

Brian (BB) King - @BBhacKing

1

1

Day Three

- 3

2

2

Day 2 Recap

- Perils of “hidden” content
- Weaknesses of client-side controls
- Common errors in filtering
- Insecure Direct Object References
- Identifiers in user-controllable places

3

3

Lab Reviews

- Bypassing the redirect filter (two ways)
- Forging a product review

4

4

Encoding Information

There is no communication without encoding.

5

5

Information

must be

ENCODED

in order to be

COMMUNICATED

6

6

Encodings

- Sound
  - Laughter, applause, a yell: humans understand
- Spoken words (in a language)
  - Some will understand, others think they do, others know they don't.
- Written words (in a language)
  - Same.
- Poetry (in a language)
  - Same. But far more room for interpretation.

7

7

Encoding for Computers

- It's all ones and zeros.
- Interpretation is what gives meaning.
- Encoding determines interpretation.

Encoding Determines Meaning

8

8

Equivocation

Letters:

```
'A' = 01000001 = 101(octal) =  65 (decimal) = 41 (hexadecimal)
'A' = 11000001 = 301(octal) = 193 (decimal) = C1 (hexadecimal)
```

Top: ASCII

Bottom: EBCIDIC

9

9

How Many People?

There are 10 kinds of people in the world:

Those who understand binary...

and

Those who don't.

10

10

Context Determines "Correct"

ASCII: 7 bits. 2^7 possibilities = 128 ~~characters~~ ~~symbols~~ *code points*

UTF-8: First 128 UTF-8 code points are the same as ASCII

UTF-8: Up to three more bytes for other symbols - 1,112,064 symbols defined.

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF[16]	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

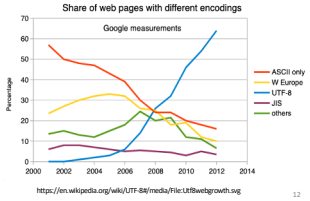
11

11

UTF-8 Is The Encoding for the Web

ASCII and UTF-8 and ISO-8859-1 (Latin-1)

...are identical for the first 128 characters.



https://en.wikipedia.org/wiki/UTF-8#/media/File:Utf8webgrowth.svg

12

12

One Step Further: Characters Mean Things

- Can't have a slash in a filename
  - Because those are directory separators
- Can't have a space in a URL
  - Because that indicates the end of the URL
- Can't have CR/LF in an HTTP header
  - Because that indicates "end of this header"
- Can't have angle brackets in HTML
  - Because those define HTML tags
- Can't have a semi-colon in a Javascript variable
  - Because those indicate "end of statement"
- Can't have a null in the middle of a C-style string
  - Because null means "end of string"

13

13

URL-Encoding

Sometimes called "Percent Encoding" (but don't do that)

Replace any "special character" with...

a percent symbol

followed by the character's 2-digit hex code.

14

14

URL-Encoding

Space becomes %20 (or +)

becomes %23

% becomes %25

& becomes %26

/ becomes %2f

: becomes %3a

= becomes %3d

. MAY become %2e

@ MAY become %40

null becomes %00

CR becomes %0d

LF becomes %0a

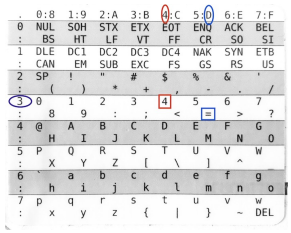
tab becomes %09

B *MAY* become %42

15

15

Cheat Sheet Sticker



.	0:8	1:9	2:A	3:B	4:C	5:D	6:E	7:F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
:	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
:	CAN	EM	SUB	EXC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'
:	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7
:	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G
:	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W
:	X	Y	Z	[	\	]	^	
6	`	a	b	c	d	e	f	g
:	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w
:	x	y	z	{	|	}	~	DEL

16

16

HTML-Encoding
(“Character Entity” Encoding)

< becomes <

> becomes >

" becomes "

' becomes '

& becomes &

> is ... what, then?

...same in XML: There are only 5 “special characters”

17

17

HTML-Encoding
(Numeric Character Reference)

Replace the character with its ~~ASCII~~ UTF-8 code point

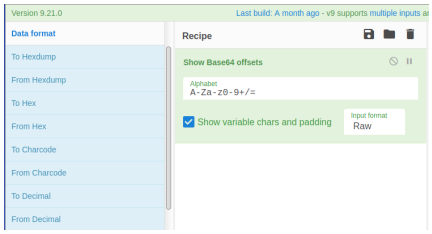
...using decimal, 'A' becomes A

...using hexadecimal, 'A' becomes A

18

18

CyberChef for Encoding and Decoding



19

19

All This Encoding Gets Complicated Fast

- Comma-Separated Values seems simple. Just add commas!
- What if a *value* has a comma in it?
  - Put quotes around the value!
- What if a *value* has quotes in it?
  - Put an extra quote in front of each one!

```
First, Last, Nickname
William, “Brooks, Sr”, ““Buffalo Bill””
```

20

20

Totally Solve-able

- But the successful solution is not the obvious solution.

21

21

Character Encoding and Filters

- Different systems and different protocols: different rules.
  - Remember: & could be %26 in a URL but & or & in HTML
- Information passes from one system to another.
  - Encoded or decoded along the way
- Pass a URL as a parameter in a URL (as in a redirect)?
  - That parameter should be URL-encoded
    - ?image_url=http%3A%2F%2Fexample%2Ecom%2Fimage%2Egif

22

22

Now You Can Answer This Question

Cross Site Scripting Prevention Search

Why Can't I Just HTML Entity Encode Untrusted Data ¶

HTML entity encoding is okay for untrusted data that you put in the body of the HTML document, such as inside a `<div>` tag. It even sort of works for untrusted data that goes into attributes, particularly if you're religious about using quotes around your attributes. But HTML entity encoding doesn't work if you're putting untrusted data inside a `<script>` tag anywhere, or an event handler attribute like onmouseover, or inside CSS, or in a URL. So even if you use an HTML entity encoding method everywhere, you are still most likely vulnerable to XSS. **You MUST use the escape syntax for the part of the HTML document you're putting untrusted data into.** That's what the rules below are all about.

23

23

Example: Lexer/Parser and Encoding

```
<!DOCTYPE HTML><HTML lang="en-us">
  <HEAD>
    <TITLE>Example</TITLE>
    <STYLE>h1 {color: #0000ff}</STYLE>
</HEAD><BODY>
  <H1>Example</H1>
  <script>alert("Example!");</script>
  <a href="https://www.google.com/search?q=parser%20lexer">
    parser&#x20;lexer
  </a>
</BODY></HTML>
```

https://validator.w3.org/nu/#textarea

24

24

Passing Values and Trusted Systems

A "Trusted System" is not the same as a "trustworthy" system.

A "Trusted System" is one you've *chosen not to defend yourself against*.

Should the back end "trust" what comes from the web server?

25

25

Lab #9: Bypass Filters to Download Forbidden Files

26

26

Lab #9 Hint

Start at /ftp/ and try to download package.json.bak or eastere.gg

Determine what the filter looks for.

Consider multiple systems may be processing the filename string.

27

27

Lab #9 *Complete:*
Bypass Filters to
Download Forbidden Files

28

28

What's Your
Takeaway from
the Lab?

29

29

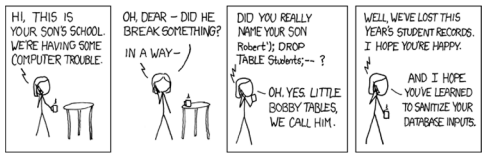
SQL Injection

As a *kind* of command injection

30

30

Can't Not Cover SQL Injection. Can't Not Show This Comic.



Exploits of a Mom: https://xkcd.com/327/

31

31

Why Does That Work?

32

32

SQL Queries: Intended Behavior

Input:
```
$name  = "Bob";
```

Query:
```
SELECT * FROM Students
WHERE (grade=11 and name='$name')
```

Result:
```
SELECT * FROM Students
WHERE (grade=11 and name='Bob');
```

33

33

SQL Injection: Unintended Behavior

```
Input:   $name  = "Robert');DROP TABLE Students; -- ";

         SELECT * FROM Students
Query:   WHERE (grade=11 and name='$name');

         SELECT * FROM Students
Result:  WHERE (grade=11 and name='Robert');DROP TABLE
         Students; -- ;')
```

34

34

Detecting SQL Injection Candidates

- Submit a string that contains "SQL Special Characters"
  - Double Quotes
  - Single Quotes (apostrophes)
  - Backticks
- Maybe an error
- More likely, a different response

35

35

Confirming SQL Injection Candidates

- Hypothesize what the query might look like
- Type that out where you can refer to it
- Submit strings that would change the behavior if they worked

...repeat.

36

36

Tools To Help

- Any webapp vulnerability scanner
- sqlmap

37

37

Possible Query for Login

```
SELECT *
FROM   users
WHERE  user='$user'
and    password='$password';
```

38

38

Lab #10:
Login Bypass by SQLi

39

39

Hints

Start simple: how might you cause a syntax error in ANY query that handles input unsafely?

What's your guess at what the query looks like in the code?

What does the Network tab (or Burp) show you?

What's the easiest way to make the query return "true"?

40

40

Lab #10 *Complete:* Login Bypass by SQLi

41

41

What's Your Takeaway from the Lab?

42

42

Credential Attacks

Gaining unauthorized access

43

43

Passwords Are Awful

- People need too many
- They're hard to keep track of without re-using
- Default credentials are still used
- Usernames are often not secrets
- Know the username? You know half of what you need...

44

44

Password Guessing

- Find a username, start guessing passwords
- But only a couple per account.
- Too many failed attempts may...
  - cause lockout
  - trigger alerts
  - get your IP address blocked

45

45

Password Spraying

- Try a likely password across all accounts you know.
- One failed logon attempt won't be suspicious.
- Given enough users, somebody's using Autumn2020!

46

46

Default Credentials

- Find a well-known account, try well-known passwords.
- admin and blank
- root and blank
- admin and password
- manager and manager
- guest and guest
- anyUsername and anyUsername1
- Any User and SeasonYear (e.g. Fall2020)

47

47

Lab #11:
Log In As the Administrator

48

48

Lab 11 Hints

Look at the reviews: find a username that might be an administrator.

Try to log in as that user. Guess at passwords.

What do you know about the password policy?

Don't waste guesses on invalid passwords.

49

49

Lab #11 *Complete:* Log In As the Administrator

50

50

Lame Attack?

51

51

52

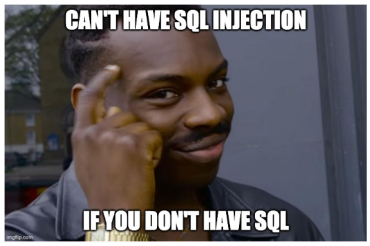
NoSQL Injection

Another kind of injection

53

53

NoSQL Doesn't Mean No Injection



54

54

NoSQL is Not a Language

- NoSQL databases are alike only in only two ways:
  - They're not relational
  - They don't use SQL
- MongoDB
- Redis
- Neo4J

Types and examples [edit]

There are various ways to classify NoSQL databases, with different categories and subcategories, some of which overlap. What follows is a basic classification by data model, with examples:

- Wide column: Accumulo, Cassandra, Scylla, HBase.
- Document: Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, eXist-db, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
- Key–value: Aerospike, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, Dynamo, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, SciDB, SDBM/Flat File dbm, ZooKeeper
- Graph: AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso

55

55

So ... "NoSQL Injection" is A Thing…
It's just not *A Thing*

- ...Not the way SQL Injection or Command Injection are, anyhow
- Each database is different
- MongoDB
  - Because it's common in webapps
    - Because it's Javascript
      - And who doesn't love more Javascript?

56

56

Javascript Outside the Browser

- Node.js: Server-side Javascript
- MongoDB: A Javascript database
- Entire applications in Javascript

57

57

Sample MongoDB Statements: Insert

```
db.inventory.insertMany([
   { item: "journal", qty: 25, status: "A",
      size: { h: 14, w: 21, uom: "cm" },
      tags: [ "blank", "red" ]
   }, . . .])
```

Similar SQL:

INSERT INTO inventory (item, qty, status…) VALUES ("journal", 25, "A"…)

58

58

MongoDB vs SQL

MongoDB	**SQL**
db.inventory.find({});	SELECT * FROM inventory;

59

59

MongoDB vs SQL

MongoDB	**SQL**
db.inventory.find({});	SELECT * FROM inventory;
db.inventory.find({ status: 'D' });	SELECT * FROM inventory WHERE status = 'D';

60

60

MongoDB vs SQL

MongoDB	SQL
db.inventory.find({});	SELECT * FROM inventory;
db.inventory.find({ status: 'D' });	SELECT * FROM inventory WHERE status = 'D';
db.inventory.find({ status: {"$ne": 'D' } });	SELECT * FROM inventory WHERE status != 'D';

61

61

MongoDB vs SQL

MongoDB	SQL
db.inventory.find({ status: {"$ne": 'D' } });	SELECT * FROM inventory WHERE status != 'D';
db.inventory.find({ status: {"$nin": "" } });	SELECT * FROM inventory WHERE status NOT NULL;

62

62

Triggering Errors

For MongoDB, any of these six "special characters" may trigger an error.

' " \ ; { }

A bigger list:

https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/Databases/NoSQL.txt

MongoDB injection reference:

https://medium.com/@fiddlycookie/nosql-injection-8732c2140576

63

63

In-Browser Server-Side Javascript Database (??)

Learning MongoDB from MongoDB, together.

Do steps 0 - 4 here:

https://docs.mongodb.com/manual/tutorial/getting-started/

64

64

65

65

Lab #12:
Mess Up The Database
With NoSQL Injection

66

66

Lab #12 *Complete:*
Mess Up The Database
With NoSQL Injection

67

67

What's Your
Takeaway from
the Lab?

68

68

Day 3 Recap

- Encoding information: context matters
- SQL Injection: less common, still a great example of injection
- Credential attacks: more about policy than web dev, but still
- NoSQL doesn't mean No Injection

69

69