Day 3: Part 1: OS cmd injection

- defn of cmd injection
- value from input to build commans that are run on the OS.
- ↳ Input vals not whitelisted
  → Account running web services is not restricted to minimum set of cmds / privs.
- This cmd inj can be done using anything (XSS, SQLi, XXE)

CmdInj → Blind
→ Non Blind (visible results)

separators (; | || & && > >>)

substitute: ` $ ( )

cat vs type; ping -c vs ping -n; ls vs dir.

- Exploit simple cmd → Review sys configs + apps → persistence or alt shell.
- Blind Injection → ICMP; DNS
  → pause using ping 127.0.0.1
  → collaborator DNS pings.

DNS → less likely filtered (compared to ICMP echo)
→ works using DNS forwarders (no direct internet required)

Day 3: Part 3: File Inclusion + Dir Traversal

- Allows attacker to include a file (L/R) due to "dynamic file inclusion" mechanism.

① LFI :→ Local file inclusion
→ retrieves files locally from ws hosting WA

look for
- → Templates
- → Docs / img
- → Framed content.

Impact:
Access to public data / code Execution.

② Dir Traversal :

- use LFI to access files outside WEB ROOT
- account running WS must have privs.
- using Relative or Absolute path references.

LFI Impact Sources
- → App config files
- → sys " "
- → source code, event logs,
- → User config files, user docs,
- → standard in file Handle.

③ RFI: retrieve files stored on remote systems across networks.

→ once LFI → check RFI
↳ proxy function to retrieve files on int n/w
RFI → Potential RCE.

Day 3: Part 5: Insec Deserialize

- AppSecCali 2015.

Serial => Variables → Bytes → Memory → Transfer
Deserial => Bytes → Variables

- can be in any OOP lang (mostly in JAVA).
- also works in .NET, PHP, Ruby, Python.

why's? ↳ easy manipulation
↳ easy maintain/debug.

Main Issue → Input Validation

Tech relying on's? ↳ RMI (Remote Method Invocation)
↳ Java Mng Extension (JMX)
↳ JMS (Java Msg Service)
↳ JSF.

① Exploit by mod sensitive params

② RCE from Insec Deserial

↳ execute unexpected code when deserial object

↳ exploited in Java apps.

↳ Deserialize, JVM calls readObject() fnc

↳ Object member fields = INPUT = attacker ctrl

↳ Any s class that targets classloader can locate+load can get deserialized.

YSOSERIAL → ↑ gadgets

↳ Allow writing files/execute processes

↳ Many found in std libraries

= collection of gadget chains in Java libs and program used to produce exploits.

Day 3 : Part 7 : SQLi Primer

- Most well known.
- Easy to address and fix (from app sec POV)
- SQL used to CRUD data from users on website
- user input → SQL Query = sql injection flaw.
- usually happens in RDBMS.
- which RDBMS? matters a lot.

SQL verbs : Select, Insert, Update, Delete, Drop, union.

SQL Query Mods :
- WHERE
- AND/OR
- LIMIT
- ORDER BY

SQL Data Types : bool, int, char, varchar, binary (vary across the RDBMS providers)

SQL injection example code:

```
1or 1=1;--
```

→ SQLi Balancing Act:

→ (prefix + payload + suffix) to exploit input flaws and cause impact

→ find REUSABLE pieces of the injection

→ requires balancing quotes to use with string data

a) Balance Quotes

b) Balance columns

c) Balance D Types

Day 3: Part 8: DISCOVERING SQLI

1) Input locations
   - GET URL
   - POST payload
   - HTTP cookie
   - HTTP UA

   } mostly blind

2) Classes of SQLi:
   - VISIBLE (Inband / Inline)
   - BLIND (OOB)

a) Inband:
- Simpler to discover
- Easier to exploit

b) OOB:
- doesn't provide visible results.
- Errors may be inline, data not
- Data " , errors not.

c) If DB error msg is seen => Not Blind SQL Inject

1st step: Check DB error → DB info
↳ ease of discovery
↳ efficacy of technique.

DB error themselves = Guide to not getting DB error msgs.
= learn from mistakes
= keep putting payloads until error changes.

d) Custom Error Messages:
→ sometimes no error msgs
→ approach a bit differently.
→ bsqli.php vs sqli.php
→ supply input as to when interpreted by backend SQL system (if present) should convert to Dent (DONT put Dent, but something converted to Dent).

→ Dent';#
Dent';--
De'/*';*/'nt
De'';'nt
De'||'nt

- Comment delimiters can allow injections to succeed that would otherwise fail.
(--, /*, */, #)
(cheap trick, dirty hack)
- used when injecting into middle of SQL query.

① Binary / Boolean inference Testing:

→ Dent' AND 1; # → Dent' AND 0; #
Dent' AND 1=1; # Dent' AND 1=0; #

② Increasing blindness (substr(L,), 1, 1) > 'a'
query.

Prefix : Dent' AND
payload : substr( ), 1, 1) > "a"
suffix : ;#

③ Blind Timing inferences:
→ inducing a delay
→ sleep(10) → MySQL
WAITFOR DELAY '0:0:10' - MSSQL

④ utter blindness: inline ⟷ OOB
↳ no err msg → no visible resp → no boolean inference opportunities without timing.

⑤ OOB Channels

- If viable, OOB = faster data exfil.
- So can use even if not needed.
- Typically leverage HTTP or DNS to tunnel comm^{n}.

Day 3: Part 9: Exploiting SQLi

① DB fingerprinting

My: 'De' 'nt', Ms: 'De'+'nt', O: 'De''nt'

Unique num functions:

My: connection_id(), Ms: @@pack_received,

O: BITAND(1,1))

② (Meta) DB info

- information_schema → ANSI SQL92 std DB provide us with relevant metadata, so no need to fingerprint at all
- not all vendors support Info-schema.
- MySQL → I_S → info of all DB
  MSSQL → I_S → info of current DB

③ DB / Tables / Columns :

MySQL vs Azure SQL vs Oracle DB

④ Exploting Inband / Inline :

- see visible results
- do more than just SELECT FROM

→ Stacked queries

⑤ Stacked Queries :

- allows to run multiple SQL queries
- split with ;
- are they supported ? mostly in MS SQL server

MySQL : supports but muddier due to application interface

Oracle : Mostly not supported.

⑥ Why stacking matters ?

→ not usually required for data exfil

→ where we want to do more than just subvert the logic of SQL, we use SQ

⑦ SQL UNION :

→ move beyond confines of the table

→ allow to access arbitrary data from dB

→ pull USERS + CUSTOMERS table

UNION Preregs:

- no. of columns pulled = original SELECT
- column data type = compatible
- know what specific tables to target.

FROMless SELECT:

→ select can be without FROM
→ Oracle needs FROM ~~for~~ all selects but has default table DUAL, to help.

Power of NULL:

→ use null to determine number of columns
→ UNION needs compat data types, NULL is compatible with all.
→ UNION + NULL = no. of columns determined.

eg:
Dent' UNION SELECT NULL, NULL, NULL, NULL; --
4 columns.

UNION + data exfil:

- once column number we get, iterate through all columns to find data exfil.

Blind Data Exfil: using Binary Inference, tread slowly through the DB to exfil data.

→ use sqlmap.

→ Inference Technique:

- TRUE = Dent Into, False = employee not found.
  - SQLi Prefix = Dent 'AND
  - SQLi suffix = ;#

Beyond DB exfiltration:

- stealing data from DB = $$ loss
- but what if org doesn't care about data exfil
- then we leverage SQLi to something else.
  - L> Delete / alter valuable data
  - L> Inject data used as stored XSS
  - L> DB priv esc (mysql) (MSSQL)
  - L> Read files (LOAD_FILE(), BULK INSERT)
  - L> write files (INTO OUTFILE)
  - L> OS interaction using Stored Procedures.

- **SQLi → write file → shell file.**
  - L> DB server also runs a web server
  - L> " " has write privs to web root
  - L> ability to browse into web root

SQLmap Tool:

-> python based CLI tool. (Bernardo Damele AG)

-> Metasploit ⟷ sqlmap.
wsab, Burp, Zap.

Day 3: Part 13: XXE

① XML used for desktop publishing, now used for data exchange.

② similar to HTML

③ derived from SGML (based on IBM's GML) dated in 1960's

④ XML used generally instead of HTML, when data sent from one app to another.

⑤ XXE = XML xternal Entity. (added in 2017).

Why XXE Matters?
- class of injection attacks
- supports EE by default (similar to LFI / RFI)
  - XXE = web app turns into a proxy, serving local + remote content.

SSRF:

→ XXE is example of SSRF
→ SSRF ~ CSRF (SSRF is server, CSRF is from client)
→ XML injection example.

→ XXE ⇒ usually flying blind
→ server to server (browser/client does not see)
→ triggered via PHP, ss request.
Challenge → inferring XXE content/code exists without direct evidence.
→ ∴ Full knowledge tests > 0 knowledge test

Blind XXE example → entry.html → entry.php → xml.php.
(client cant see/interpret req).