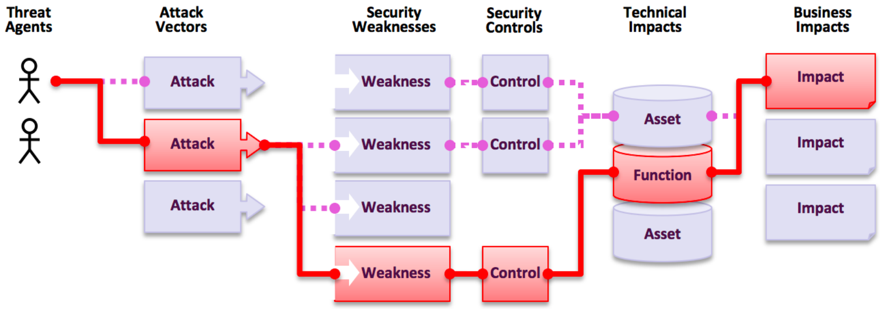
# OWASP Top Ten Part Two

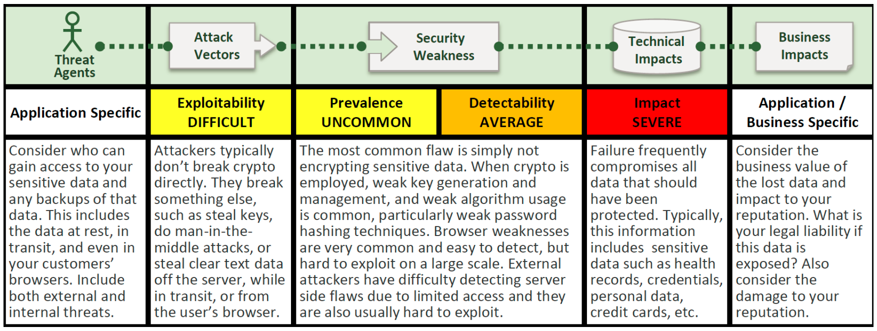
Laurie Williams
lawilli3@ncsu.edu
John Slankas
jbslanka@ncsu.edu

## Application Security Risk



file:///Users/lawilli3/Downloads/OWASP%20Top%2010%20-%202017%20RC1-English%20(1).pdf

# A6: Sensitive Data Exposure



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>DIFFICULT | Prevalence<br>UNCOMMON | Detectability<br>AVERAGE | Impact<br>SEVERE | Application /<br>Business Specific |
| Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats. | Attackers typically don't break crypto directly. They break something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser. | The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit. | | Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc. | Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also consider the damage to your reputation. |

# What is Sensitive Data?

- Data Classification Guidelines
- SP 800-60: Guidelines for Mapping Types of Information and Information Systems to Security Categories

## Example Set

- Sensitive: Security Q&As, passwords, keys
- Confidential: SSN, Account Numbers, Driver's license, protected health information
- Company: transactions, corporate directory
- Public: Job postings, products

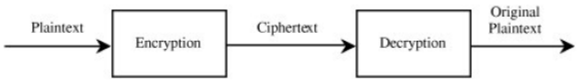
## Another Example Set

| Category | Examples |
|---|---|
| Prohibited | Social Security Numbers, Credit Card Numbers, |
| Restricted | Health Information, Passport Numbers, |
| Confidential | Student Records, Employment Applications, Contracts, University and Employee IDs |
| Unrestricted | SUNet (Unity) IDs, published research data, directory |

http://web.stanford.edu/group/security/securecomputing/dataclass_chart.html

## Encryption

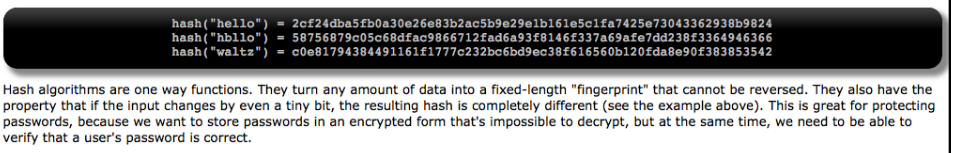
- **Encryption** is the process of encoding a message/data so its meaning is not obvious
- **Decryption** is the reverse process, transforming an encrypted message/data back into its normal, original form
- **Plaintext**: original message/data
- **Ciphertext:** encrypted message/data



Pfleeger and Pfleeger, *Security in Computing*, Pearson Education, 2003.

## Hash Algorithms

- Provide a mapping between an arbitrary length input, and a (usually) fixed length (or smaller length) output.
- Use a hash function when you want to compare a value but can't store the plain representation (for any number of reasons) … and don't necessarily need the input data back.
  - Passwords should fit this use-case very well since you don't want to store them plain-text for security reasons (and shouldn't).
- Example hash algorithms: crc32,  MD5, SHA-1, SHA-256



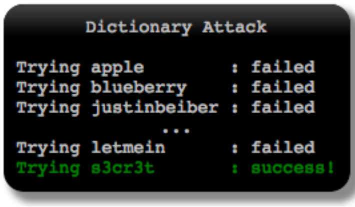
```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366
hash("waltz") = c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542
```

Hash algorithms are one way functions. They turn any amount of data into a fixed-length "fingerprint" that cannot be reversed. They also have the property that if the input changes by even a tiny bit, the resulting hash is completely different (see the example above). This is great for protecting passwords, because we want to store passwords in an encrypted form that's impossible to decrypt, but at the same time, we need to be able to verify that a user's password is correct.

http://stackoverflow.com/questions/4948322/fundamental-difference-between-hashing-and-encryption-algorithms
https://crackstation.net/hashing-security.htm

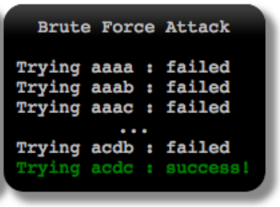
# Cracking Hash Algorithms

- Dictionary Attack: uses a file containing words, phrases, common passwords, and other strings that are likely to be used as a password.
- Brute Force Attack: tries every possible combination of characters up to a given length (computationally expensive)
- Lookup tables (e.g. rainbow tables) are pre-computed



```
            Dictionary Attack

Trying apple        : failed
Trying blueberry    : failed
Trying justinbeiber : failed
             ...
Trying letmein      : failed
Trying s3cr3t       : success!
```



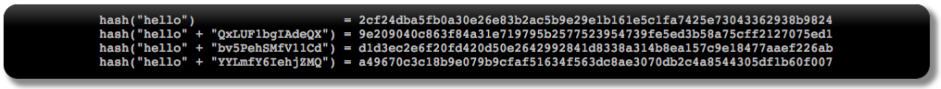
```
     Brute Force Attack

Trying aaaa : failed
Trying aaab : failed
Trying aaac : failed
       ...
Trying acdb : failed
Trying acdc : success!
```

https://crackstation.net/hashing-security.htm

# Adding Salts with Hashing

- Add a salt (a random value ) to the value to be hashed. The combined value is hashed, and often the salt is then stored in plain text along with the hash value. If the hashed value needs to be compared with input (for instance to check a password), the input to be verified is combined once again with the salt and then the hash is compared to the stored hash.
- A new random salt must be generated each time a user creates an account or changes their password.
- Salt should be generated using a **Cryptographically Secure Pseudo-Random Number Generator** (CSPRNG)



```
hash("hello")                    = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hello" + "QxLUF1bgIAdeQX") = 9e209040c863f84a31e719795b2577523954739fe5ed3b58a75cff2127075ed1
hash("hello" + "bv5PehSMfV11Cd") = d1d3ec2e6f20fd420d50e2642992841d8338a314b8ea157c9e18477aaef226ab
hash("hello" + "YYLmfY6IehjZMQ") = a49670c3c18b9e079b9cfaf51634f563dc8ae3070db2c4a8544305df1b60f007
```

https://crackstation.net/hashing-security.htm

# Transport Layer Protection

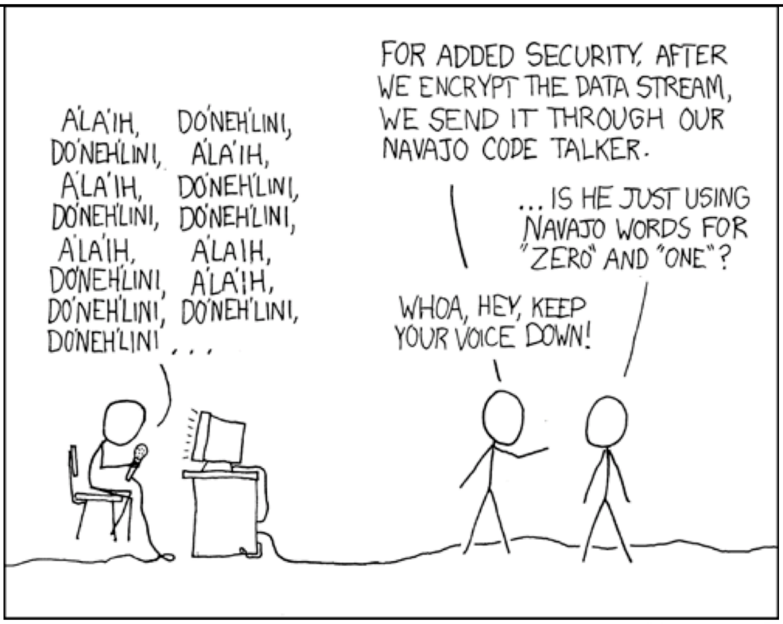
- Network encryption protocol (such as IPSec or SSL) protects data in transit.
- If you have SSL enabled for the webserver (requires dedicated IP) adding https:// to the url will encrypt the connection and whatever page the url points to will be encrypted while *in transit* … can put secure pages in a subdomain
- Set the secure flag to protect a session ID or authentication token … or attacker can impersonate the victim by sniffing session ID
- Provides endpoint authentication (client optional)
- Encrypts all communication
- Uses both asymmetric and symmetric algorithms
  - Use strong encryption algorithms
- For web applications with a need for security, use SSL for the entire session (login to logout)

# Example Attacks

- **Scenario #1:** A site simply doesn't use SSL for all authenticated pages. Attacker simply monitors network traffic (like an open wireless network), and steals the user's session cookie. Attacker then replays this cookie and hijacks the user's session, accessing the user's private data.

- **Scenario #3:** The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of precalculated hashes.

https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure

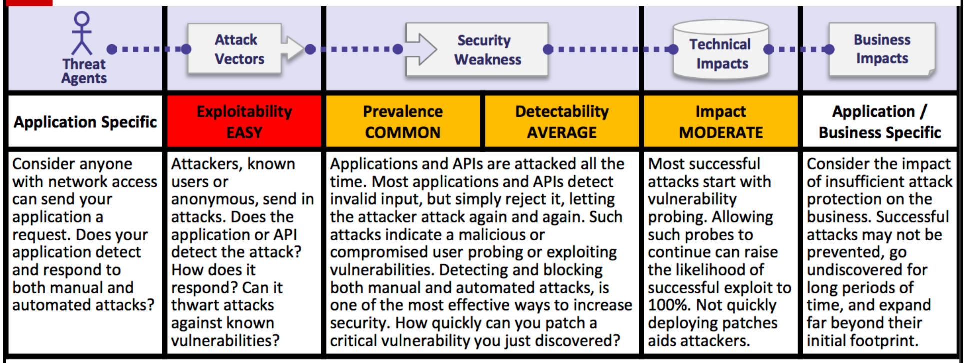

ALA'IH, DO'NEH'LINI,
DO'NEH'LINI, ALA'IH,
ALA'IH, DO'NEH'LINI,
DO'NEH'LINI, DO'NEH'LINI,
ALAIH, ALAIH,
DO'NEH'LINI, ALA'IH,
DO'NEH'LINI, DO'NEH'LINI,
DO'NEH'LINI . . .
FOR ADDED SECURITY, AFTER WE ENCRYPT THE DATA STREAM, WE SEND IT THROUGH OUR NAVAJO CODE TALKER.
... IS HE JUST USING NAVAJO WORDS FOR "ZERO" AND "ONE"?
WHOA, HEY, KEEP YOUR VOICE DOWN!

http://xkcd.com/257/

## Class Exercise

- Examine the database structure for OpenEMR: http://www.open-emr.org/wiki/index.php/Database_Structure
- List 5 fields that should be encrypted or hashed. Provide the table name, column name, whether encrypted/hashed, and a brief reason.

# A7: Insufficient Attack Protection (new)



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability EASY | Prevalence COMMON | Detectability AVERAGE | Impact MODERATE | Application / Business Specific |
| Consider anyone with network access can send your application a request. Does your application detect and respond to both manual and automated attacks? | Attackers, known users or anonymous, send in attacks. Does the application or API detect the attack? How does it respond? Can it thwart attacks against known vulnerabilities? | Applications and APIs are attacked all the time. Most applications and APIs detect invalid input, but simply reject it, letting the attacker attack again and again. Such attacks indicate a malicious or compromised user probing or exploiting vulnerabilities. Detecting and blocking both manual and automated attacks, is one of the most effective ways to increase security. How quickly can you patch a critical vulnerability you just discovered? | | Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of successful exploit to 100%. Not quickly deploying patches aids attackers. | Consider the impact of insufficient attack protection on the business. Successful attacks may not be prevented, go undiscovered for long periods of time, and expand far beyond their initial footprint. |

file:///Users/lawilli3/Downloads/OWASP%20Top%2010%20-%202017%20RC1-English%20(1).pdf

# Four strategies to handle threats - 1

- Prevent compromise
  - Use software security techniques we will learn in this class to prevent both design flaws and implementation bugs
- Detect attack (Intrusion detection)
  - Understand behavior of benevolent user. Detect if activity is underway that is causing the system to be used in an atypical way (tempo too high, atypical input, unusual usage pattern, repeated requests)

## Four strategies to handle threats - 2

- Respond to attack
  - Proactive response to logs and notifications are critical.  Decide when to automatically block requests, IP addresses or IP ranges.  Consider monitoring or disabling misbehaving user accounts
- Patch quickly
  - Develop a dev process that can push out critical patches very quickly (DevOps/continuous deployment practices helps)

Computer Science
NC STATE UNIVERSITY

## Example attack scenarios

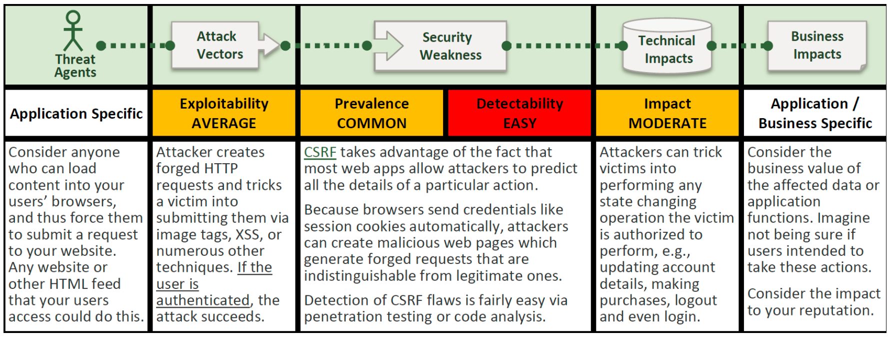
- Scenario #1: Attacker uses automated tool like OWASP ZAP or SQLMap to detect vulnerabilities and possibly exploit them. Attack detection should recognize the application is being targeted with unusual requests and high volume. **Automated scans should be easy to distinguish from normal traffic**.
- Scenario #2: A skilled human attacker carefully probes for potential vulnerabilities, eventually finding an obscure flaw. While more difficult to detect, this attack still involves requests that a normal user would never send, such as input not allowed by the UI. **Tracking this attacker may require building a case over time that demonstrates malicious intent.**
- Scenario #3: Attacker starts exploiting a vulnerability in your application that your current attack protection fails to block. How quickly can you **deploy a patch** to block continued exploitation of this vulnerability?

Computer Science
NC STATE UNIVERSITY

# Class Exercise

- What are the capabilities of OWASP ZAP? How can these capabilities be used by developers? By attackers?
- What are the capabilities of sqlmap? How can these capabilities be used by developers? By attackers?
- Find the OWASP Automated Threat Handbook Web Application. Starting on Page 26, the handbook defines automated threat events. Summarize three threat events, including the possible symptoms (that should be used for detection purposes) and three suggested countermeasures for each event.

# A8: Cross-site Request Forgery



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>COMMON | Detectability<br>EASY | Impact<br>MODERATE | Application / Business Specific |
| Consider anyone who can load content into your users' browsers, and thus force them to submit a request to your website. Any website or other HTML feed that your users access could do this. | Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS, or numerous other techniques. If the user is authenticated, the attack succeeds. | CSRF takes advantage of the fact that most web apps allow attackers to predict all the details of a particular action.<br>Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones.<br>Detection of CSRF flaws is fairly easy via penetration testing or code analysis. | | Attackers can trick victims into performing any state changing operation the victim is authorized to perform, e.g., updating account details, making purchases, logout and even login. | Consider the business value of the affected data or application functions. Imagine not being sure if users intended to take these actions.<br>Consider the impact to your reputation. |

## Cross site request forgery

- Attacker tricks a browser into performing undesired requests to websites on behalf of logged-in users
- The attack is performed by including in a page either an image (IMG tag) or an iframe pointing to a site where the user is presumed to be already logged in.
  - The sites that are more likely to be attacked are community Websites (social networking, email) or sites that have high dollar value accounts associated with them (banks, stock brokerages, bill pay services).
- If the conditions are right, the malicious request includes the victim's credentials when sent to the vulnerable site.

## Successful CSRF

- Several things have to happen for cross-site request forgery to succeed:
  - The attacker must target either a site that doesn't check the referrer header (which is common)
  - The attacker must find a form submission at the target site, or a URL that has side effects, that does something (e.g., transfers money, or changes the victim's e-mail address or password).
  - The attacker must determine the right values for all the form's or URL's inputs; if any of them are required to be secret authentication values or IDs that the attacker can't guess, the attack will fail.
  - The attacker must lure the victim to a Web page with malicious code while the victim is logged in to the target site.

http://en.wikipedia.org/wiki/Cross-site_request_forgery

# CSRF Mitigation

- Anti-CSRF token(s)
  - Nonce: one-time cryptographically random token that is returned to the client. The nonce is sent to the client and also saved on the server and compared when the action comes in.
  - HMAC: encrypted hash or "keyed hash" of the page combined with the session ID. If you create an HMAC of the Page URL, plus the User ID or Session ID a comparison value can be created that will make distributed attacks very difficult.
- Checking HTTP Referer or HTTP Origin header
- Don’t use GET parameters

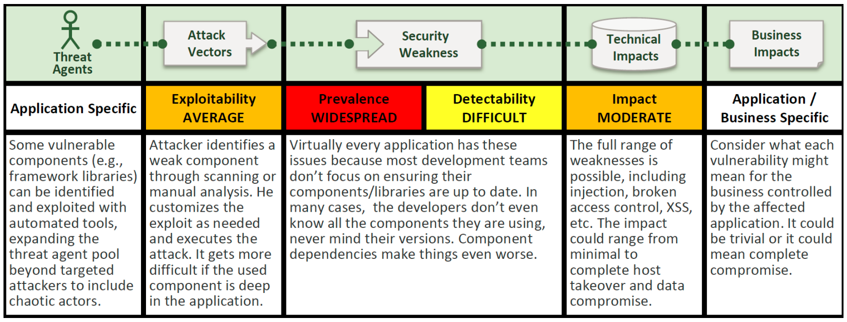
# CSRF Video

- https://www.youtube.com/watch?v=QJmYhIJraOo

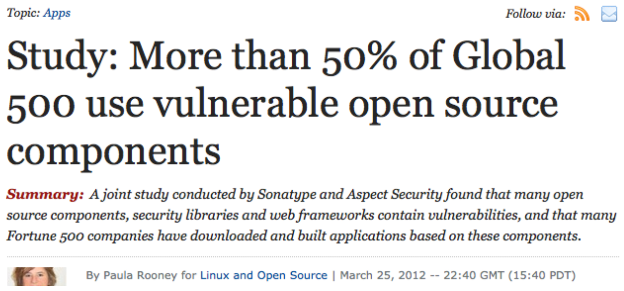
# Class Exercise

- Design and desrcirbe a CSRF attack against BodgeIT to change a user's password.

# A9: Using Components with Known Vulnerabilities



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability<br>AVERAGE | Prevalence<br>WIDESPREAD | Detectability<br>DIFFICULT | Impact<br>MODERATE | Application / Business Specific |
| Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools, expanding the threat agent pool beyond targeted attackers to include chaotic actors. | Attacker identifies a weak component through scanning or manual analysis. He customizes the exploit as needed and executes the attack. It gets more difficult if the used component is deep in the application. | Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date. In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse. | | The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could range from minimal to complete host takeover and data compromise. | Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise. |



Topic: Apps

Follow via:

# Study: More than 50% of Global 500 use vulnerable open source components

**Summary:** *A joint study conducted by Sonatype and Aspect Security found that many open source components, security libraries and web frameworks contain vulnerabilities, and that many Fortune 500 companies have downloaded and built applications based on these components.*

By Paula Rooney for Linux and Open Source | March 25, 2012 -- 22:40 GMT (15:40 PDT)

# Example Attack Scenarios

- Component vulnerabilities can cause almost any type of risk imaginable, ranging from the trivial to sophisticated malware designed to target a specific organization. Components almost always run with the full privilege of the application, so flaws in any component can be serious, The following two vulnerable components were downloaded 22m times in 2011.
  - Apache CXF Authentication Bypass – By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)
  - Spring Remote Code Execution – Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server.
- Every application using either of these vulnerable libraries is vulnerable to attack as both of these components are directly accessible by application users. Other vulnerable libraries, used deeper in an application, may be harder to exploit.

https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities

# The extent of the problem

- 35% of the average commercial application is open source
- An application has 105 open source dependencies
- Which is 2x more that the application authors were expecting
- 67% of the applications contain third-party dependency security vulnerabilities
  - 40% of them had a CVSS rating of "severe"
- 22.5 vulnerability per application
- Each of which was present there on average for 1894 days
  - Heartbleed present in 10% of applications 18 months after widespread news of the vulnerability

The State of Open Source Security in Commercial Applications

# Prevention

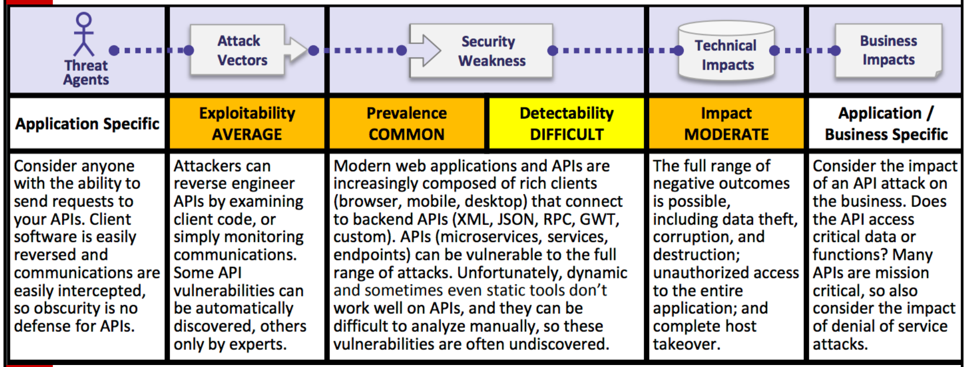
- Most component projects do not create vulnerability patches for old versions. Instead, most simply fix the problem in the next version. So upgrading to these new versions is critical. Software projects should have a process in place to:
  - Identify all components and the versions you are using, including all dependencies. (e.g., the versions plugin).
  - Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep updated.
  - Establish security policies governing component use, such as requiring certain software development practices, and passing security tests.
  - Where appropriate, consider adding security wrappers around components to disable unused functionality and/ or secure weak or vulnerable aspects of the component.
  - OWASP Dependency-Check a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities

https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities

# Exercises

- Learning about NVD, CVSS, Dependency Checker

# A10: Underprotected APIs (new)



| Threat Agents | Attack Vectors | Security Weakness | | Technical Impacts | Business Impacts |
|---|---|---|---|---|---|
| Application Specific | Exploitability AVERAGE | Prevalence COMMON | Detectability DIFFICULT | Impact MODERATE | Application / Business Specific |
| Consider anyone with the ability to send requests to your APIs. Client software is easily reversed and communications are easily intercepted, so obscurity is no defense for APIs. | Attackers can reverse engineer APIs by examining client code, or simply monitoring communications. Some API vulnerabilities can be automatically discovered, others only by experts. | Modern web applications and APIs are increasingly composed of rich clients (browser, mobile, desktop) that connect to backend APIs (XML, JSON, RPC, GWT, custom). APIs (microservices, services, endpoints) can be vulnerable to the full range of attacks. Unfortunately, dynamic and sometimes even static tools don't work well on APIs, and they can be difficult to analyze manually, so these vulnerabilities are often undiscovered. | | The full range of negative outcomes is possible, including data theft, corruption, and destruction; unauthorized access to the entire application; and complete host takeover. | Consider the impact of an API attack on the business. Does the API access critical data or functions? Many APIs are mission critical, so also consider the impact of denial of service attacks. |

file:///Users/lawilli3/Downloads/OWASP%20Top%2010%20-%202017%20RC1-English%20(1).pdf

# Example Attack Scenarios

- Scenario #1: Imagine a mobile banking app that connects to an XML API at the bank for account information and performing transactions. The attacker reverse engineers the app and discovers that the user account number is passed as part of the authentication request to the server along with the username and password. The attacker sends legitimate credentials, but another user's account number, gaining full access to the other user's account.
- Scenario #2: Imagine a public API offered by an Internet startup for automatically sending text messages. The API accepts JSON messages that contain a "transactionid" field. The API parses out this "transactionid" value as a string and concatenates it into a SQL query, without escaping or parameterizing it. As you can see the API is just as susceptible to SQL injection as any other type of application. In either of these cases, the vendor may not provide a web UI to use these services, making security testing more difficult.

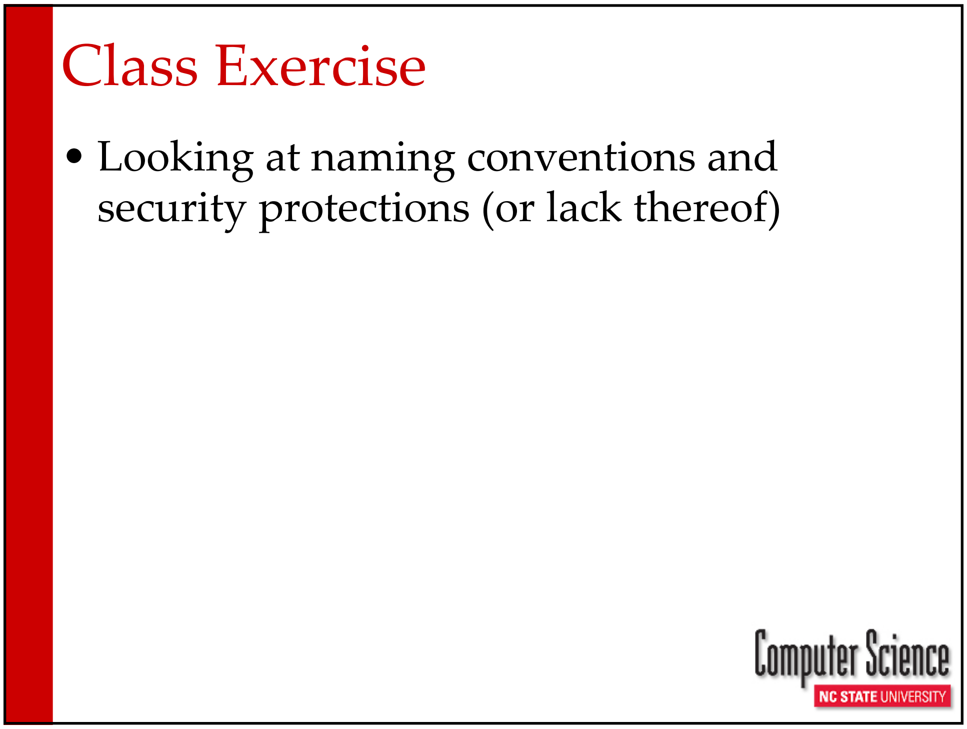
Computer Science
NC STATE UNIVERSITY

# Prevention

The key to protecting APIs is to ensure that you fully understand the threat model and what defenses you have:

- 1. Ensure that you have secured communications between the client and your APIs.
- 2. Ensure that you have a strong authentication scheme for your APIs, and that all credentials, keys, and tokens have been secured.
- 4. Implement an access control scheme that protects APIs from being improperly invoked, including unauthorized function and data references.
- 5. Protect against injection of all forms, as these attacks are just as viable through APIs as they are for normal apps.
- Be sure your security analysis and testing covers all your APIs

Computer Science
NC STATE UNIVERSITY



## Class Exercise

- Looking at naming conventions and security protections (or lack thereof)