Hochschule Konstanz
Fakultät Informatik

Software Security

AIN

Hanno Langweg

01 Software Vulnerabilities

Software vulnerabilities

- Input validation and representation
- API abuse
- Security features (absence of ~)
- Time and state => race conditions
- Error handling
- Code quality
- Encapsulation
- Environment

McGraw (2006) Software Security: Building Security In, ch. 12

CWE

- Common Weakness Enumeration, cwe.mitre.org
- Project started in 2006; goal: easier comparison and data exchange between software security tools
- 1,023 weaknesses in 237 categories
  Different views: research, development, architecture

CWE, development concepts

699 - Development Concepts

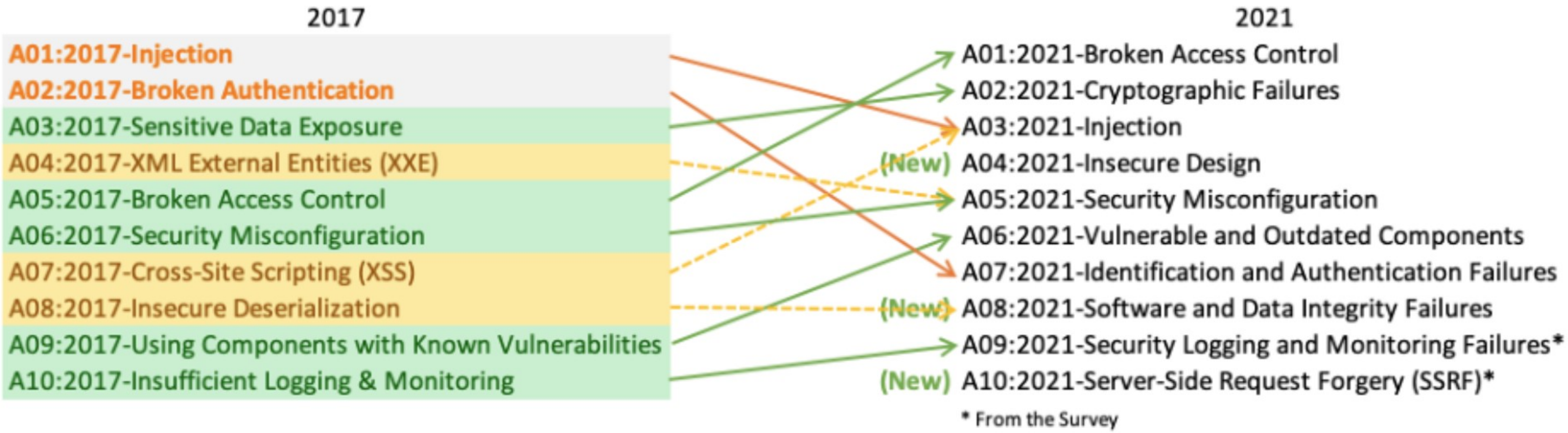
- Configuration - *(16)*
- Data Processing Errors - *(19)*
  - Improper Encoding or Escaping of Output - *(116)*
  - Improper Restriction of Operations within the Bounds of a Memory Buffer - *(119)*
  - String Errors - *(133)*
  - Type Errors - *(136)*
  - Representation Errors - *(137)*
    - Improper Neutralization of Special Elements - *(138)*
    - Cleansing, Canonicalization, and Comparison Errors - *(171)*
    - Reliance on Data/Memory Layout - *(188)*
    - Improper Handling of Syntactically Invalid Structure - *(228)*
  - Information Management Errors - *(199)*
  - Improper Input Validation - *(20)*
  - Data Structure Issues - *(461)*
  - Modification of Assumed-Immutable Data (MAID) - *(471)*
  - Automated Recognition Mechanism with Inadequate Detection or Handling of Adv
- Pathname Traversal and Equivalence Errors - *(21)*
- Numeric Errors - *(189)*
- 7PK - Security Features - *(254)*
- 7PK - Time and State - *(361)*
- Error Conditions, Return Values, Status Codes - *(389)*
- Resource Management Errors - *(399)*
- Channel and Path Errors - *(417)*

Vulnerabilities in Web Applications

OWASP Top 10

- OWASP Open Web Application Security Project
  www.owasp.org
- List of most frequently occurring types of vulnerabilities in web applications
- Updated every 3-4 years
- Changes 2017 ➔ 2021
  - More often: Broken Access Control, Security Misconfiguration, Vulnerable and Outdated Components
  - Less often: Injection, Identification and Authentication Failures

OWASP Top 10



– Note: Top of the list affects ca. 4% of applications, end of list 2% of applications ➔ all top 10 important

Top Web Application Vulnerabilities A01

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 (SSRF)*

- Access control might be present, but may not cover all access paths to an object
- A direct object reference occurs when a developer exposes a reference to an object, such as a file, directory, URL, or database key. Without an access control check, attackers can manipulate these references to access unauthorized data.
- Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar checks when these pages are accessed, or attackers might forge URLs to access these hidden pages anyway.

Top Web Application Vulnerabilities A02

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 (SSRF)*

- Many web application do not properly protect sensitive data, such as credit cards, social security numbers, and authentication credentials, with appropriate encryption or hashing.
- Attackers may use this weakly protected data to conduct identity theft, credit card fraud, or other crimes.

Top Web Application Vulnerabilities A03

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 (SSRF)*

- Injection flaws occur when untrusted data is sent to an interpreter as part of a command or query, e.g. SQL, OS (shell), LDAP injection
- Data sent by attacker interpreted as commands in application context
- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping.
- XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Top Web Application Vulnerabilities A04

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 (SSRF)*

- Missing or ineffective control design
- E.g., insufficiently protected credentials, trust boundary violations, error messages containing sensitive information
- Difference between insecure design and insecure implementation
- Threat modeling for important parts of application
- Integrate plausibility checks, unit+integration tests, limit resource consumption

Top Web Application Vulnerabilities A05

- Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform.
- All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.

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 (SSRF)*

Top Web Application Vulnerabilities A06

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 (SSRF)*

- Components, such as libraries, frameworks, and other software modules, almost always run with full privileges.
- If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.
- Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

Top Web Application Vulnerabilities A07

– Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords keys, session tokens, or exploit implementation flaws to assume other users' identities.

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 (SSRF)*

Top Web Application Vulnerabilities A08

- Integration of plugins/libraries from untrustedworthy sources, e.g., repositories, remote servers, CDNs
- Serialization/deserialization of objects
- Trustworthy repositories, digital signatures

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 (SSRF)*

Top Web Application Vulnerabilities A09

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 (SSRF)*

– Detection is necessary for reaction and learning
– Attackers might unsuccessfully try many attacks until they obtain information or succeed in changing control flow of application
– Log both successful and unsuccessful actions
– Protect integrity and authenticity of log data
– Prepare logs for automated processing and correlation

Top Web Application Vulnerabilities A10

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 (SSRF)*

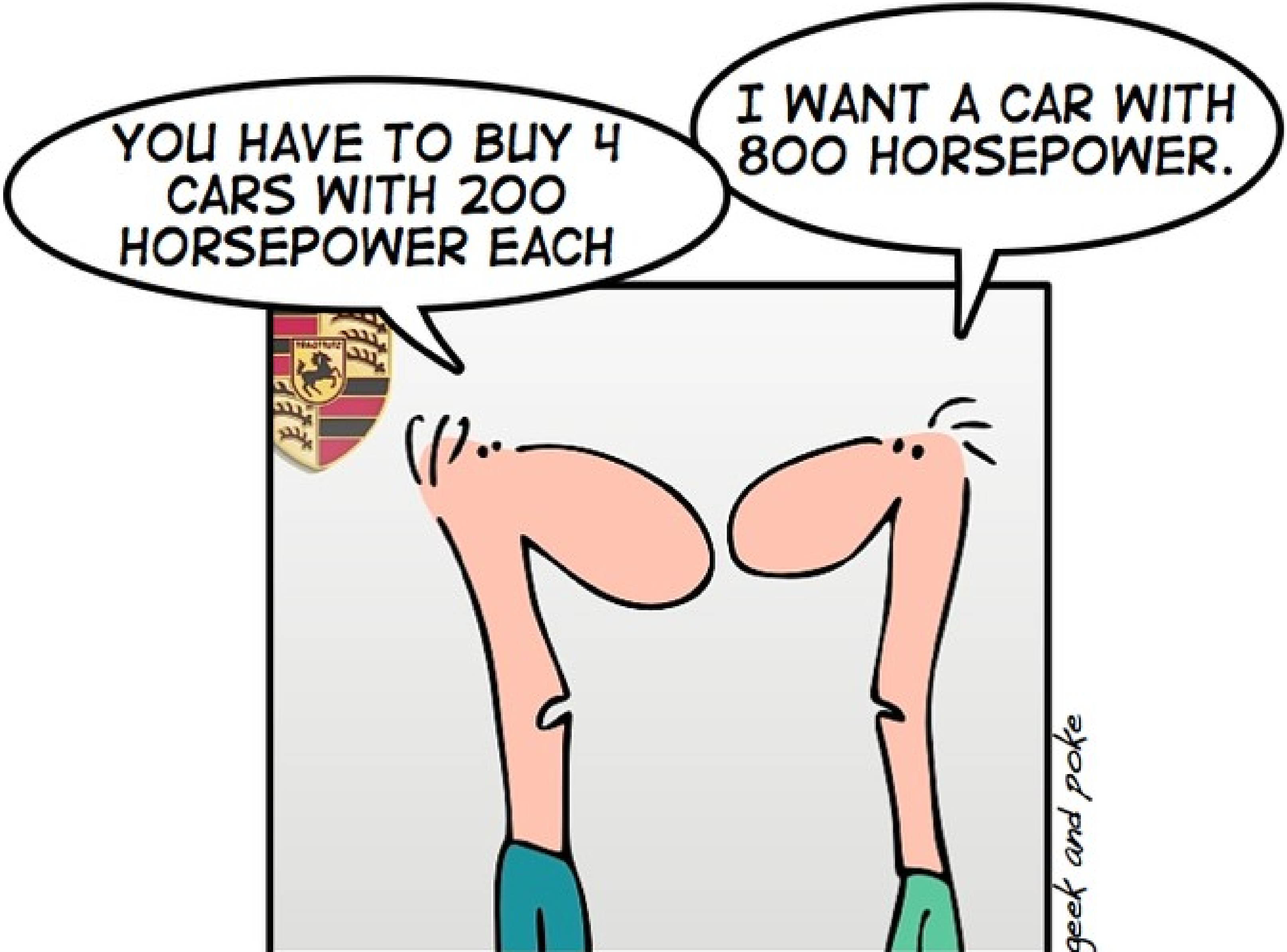
– Web application fetches remote resource without validating user-supplied URL

– Attacker circumvents restrictions for client, e.g., firewall rules by having server access resource with fewer restrictions placed on requests made by server compared to requests made by client browser

Concurrency and Race Conditions

Drivers of concurrency

- **Asynchronous** events (interrupts, input, network traffic)
- Multitasking, **multithreading**
- **Client/server** architectures
- **Multicore** hardware
- **Cloud** computing



THE AUTOMOTIVE INDUSTRY HAS ADOPTED THE STRATEGY FROM THE PROCESSOR MANUFACTURERS

http://geek-and-poke.com/?offset=1213057560000

«**Concurrency** - Execution of more than one procedure at the same time (perhaps with the access of shared data), either truly simultaneously (as on a multiprocessor) or in an unpredictably interleaved manner.»

— Encyclopædia Britannica

Race condition

– **Simultaneous operation** on **same resource** leading to **nondeterministic** computation
– Unexpected sequence of events
– Violating properties of a transaction

– History: timing problems in electronic circuits

https://en.wikipedia.org/wiki/Formula_racing#/media/File:First_lap_2014_Bahrain_Grand_Prix_(3).jpg

TOCTTOU

- Time of check to time of use
  - Change of resource, reference, or subject between privilege check and resource access
  - Security state is not maintained
- Exploitation requires exact timing

TOCTTOU example

- Transportation
- Time of **check**: after boarding the vehicle
- Time of **use**: ride
- Counter measures:
  - Periodic ticket controls (limited window of opportunity)
  - Passenger-destination memorization ("session state")
  - Exit controls (recovery)

https://inside.bahn.de/einfach-buchen-onlinetickets/

TOCTTOU example

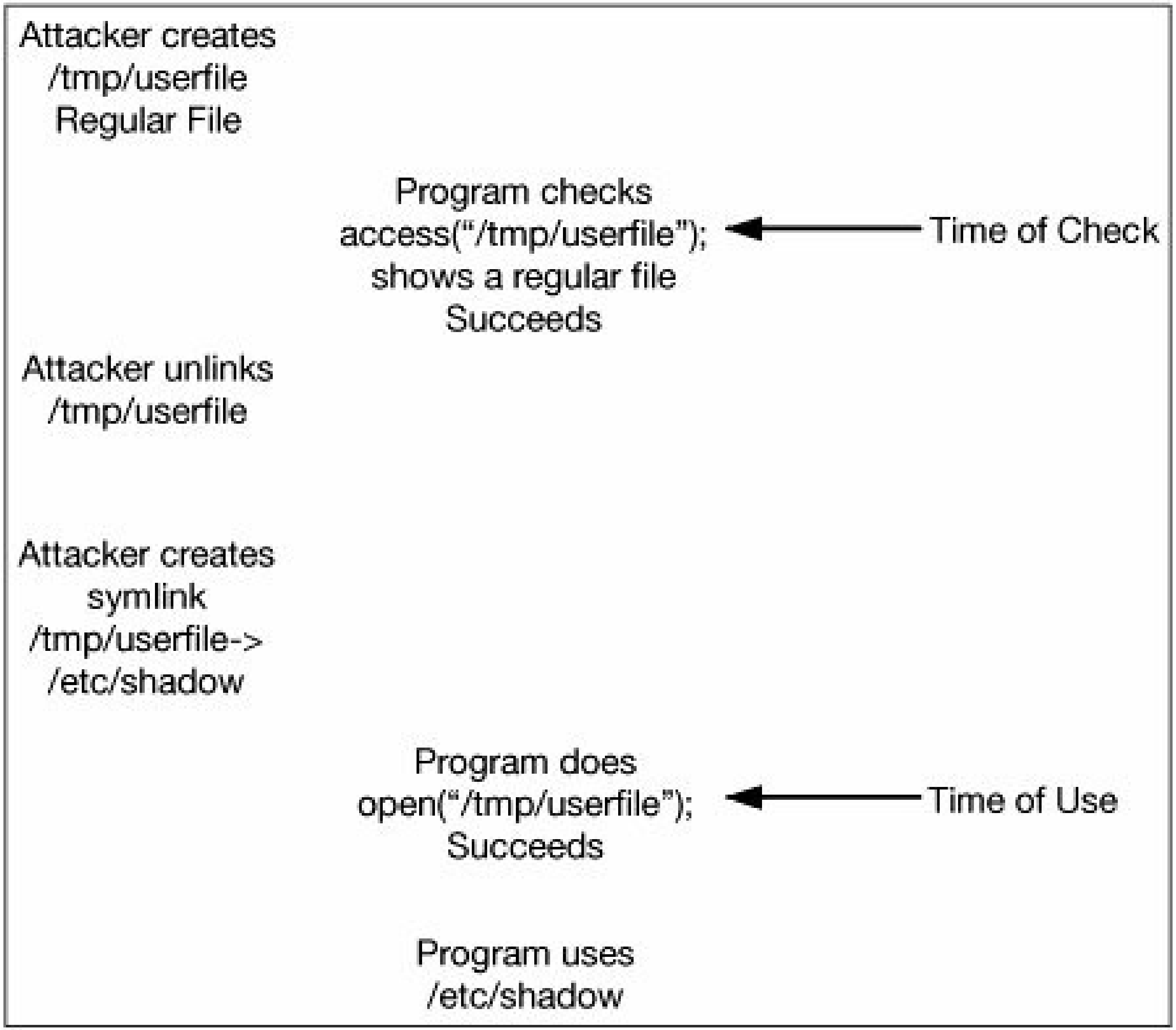
– Unix file access

```
res = access("/tmp/userfile", R_OK);
if (res!=0)
      die("access");
/* Ok, we can read from /tmp/userfile */
fd = open("/tmp/userfile", O_RDONLY);
```

TOCTTOU example

– Unix file access

```
res = access("/tmp/userfile", R_OK);
if (res!=0)
      die("access");
/* Ok, we can read from /tmp/userfile */
/* What can happen between access() and open()? */
fd = open("/tmp/userfile", O_RDONLY);
/* And what would be the effect? */
```



Dowd et al. (200x). The Art of Software Security Assessment. Fig. 9-7

Typical suspicious functions

- **access()** checks for access rights on a file, but file might be altered after check
- **stat()** provides information about a file, but file might be altered after request
- Both functions reference by file name
  - Alternative to access(): attempt to use file and handle failure to open the file
  - Alternative to stat(): fstat() uses file descriptor

Permission race condition

```
FILE *fp;
int fd;
if (!(fp=fopen(myfile, "w+")))
   die("fopen");

/* we'll use fchmod() to prevent a race condition
*/
fd=fileno(fp);
/* let's modify the permissions */
if (fchmod(fd, 0600)==-1)
   die("fchmod");
```

Permission race condition

```
FILE *fp;
int fd;
if (!(fp=fopen(myfile, "w+")))
   die("fopen");
/* File might have just been created
   with standard permissions */
/* we'll use fchmod() to prevent a race condition
*/
fd=fileno(fp);
/* let's modify the permissions */
if (fchmod(fd, 0600)==-1)
   die("fchmod");
```

Temporary files

- Many processes access temp directory
- Unique files
  - Prevent others from having access to file
  - Access file created in the past by same program
- Need to check that other processes do not have access
- Unix: Use O_TMPFILE option with open()/openat() calls

Source: http://out-println.appspot.com/posts/tocttou

TOCTTOU Java sample

```java
import java.util.Date;

/** An immutable class representing a time interval. */
public final class Interval {

  private final Date min;
  private final Date max;

  public Interval(Date min, Date max) {
    if (min.after(max)) throw new IllegalArgumentException();
    this.min = (Date) min.clone();
    this.max = (Date) max.clone();
  }

  public Date min() { return (Date) min.clone(); }
  public Date max() { return (Date) max.clone(); }
}
```

Create an instance of *Interval* with min > max?

TOCTTOU Java sample

```java
public class Attacker {
  volatile Date max = new Date();

  final Thread burglar = new Thread() {
    @Override public void run() {
      max.setTime(0);
    }
  };
```

Create an instance of *Interval* with min > max?

TOCTTOU Java sample

```java
    public Interval attack() {
        Date min = new Date();
        max.setTime(min.getTime());
        burglar.start();
        try {
            Interval i = new Interval(min, max);
            if (i.min().after(i.max())) {
                System.out.println("Success!");
                return i;
            } else System.out.println("Too late");
        }
        catch (final IllegalArgumentException e) {
            System.out.println("Too soon");
        }
        return null;
    }
}
```

Create an instance of *Interval* with min > max?

TOCTTOU Java sample

```java
Interval corrupted;
long start = System.currentTimeMillis();
int count = 0;
while ((corrupted = (new Attacker()).attack()) == null) count += 1;

System.out.println("In " + (System.currentTimeMillis() - start) + " ms and " +
  count + " attacks, I have created a corrupt interval with min=" +
  corrupted.min() + " and max=" + corrupted.max());
```

```
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Too late
Success!
In 11574 ms and 72922 attacks, I have created a corrupt interval with min=Wed Se
p 25 09:03:30 CEST 2013 and max=Thu Jan 01 01:00:00 CET 1970
```

Create an instance of *Interval* with min > max?

TOCTTOU Java sample

```java
import java.util.Date;

/** An immutable class representing a time interval. */
public final class Interval {

  private final Date min;
  private final Date max;

  public Interval(Date min, Date max) {
    if (min.after(max)) throw new IllegalArgumentException();
    this.min = (Date) min.clone();
    this.max = (Date) max.clone();
  }

  public Date min() { return (Date) min.clone(); }
  public Date max() { return (Date) max.clone(); }
}
```

Create an instance of *Interval* with min > max?

TOCTTOU Java sample

```java
import java.util.Date;

/** An immutable class representing a time interval. */
public final class Interval {

  private final Date min;
  private final Date max;

 public Interval(Date min, Date max) {
   this.min = (Date) min.clone();
   this.max = (Date) max.clone();
   if (this.min.after(this.max)) throw new IllegalArgumentException();
 }
  }

  public Date min() { return (Date) min.clone(); }
  public Date max() { return (Date) max.clone(); }
}
```

Create an instance of *Interval* with min > max?

Prevention

- Confirm resource, reference, subject identity upon access
  - If not possible to prevent, limit window of opportunity
  - If not possible to prevent, log operations

Summary

- Software vulnerabilities
  - Defensive programming, untrusted input, architectural design principles
  - CWE as an attempt to systematically discuss vulnerabilities
- OWASP Top 10
  - Typical vulnerabilities in web applications; reachable by many attackers, large attack surface
  - Progress with elimination of vulnerabilities like cross-site scripting or cross-site request forgery owing to protection mechanisms included in web application frameworks
- Race conditions
  - Hard to discover, hard to exploit