Secure Coding Fundamentals

Amt der Niederösterreichischen Landesregierung, 11.12.2025

Ulrich Bayer, SBA Research

About SBA Research and Me

SBA Research

- Research Center for Information Security
- ~120 employees

Dr. Ulrich Bayer

- Studied @ Vienna University of Technology
- Worked as a software developer
- Security Consultant since 2010
- Team Lead Software Security Group

Training “Rules”

- Feel free to interrupt! Please ask questions any time. We have lots of content to cover but also time for discussions and to answer questions

- If you have made a different experience or have another opinion, please share it. Let’s discuss it – Security is rarely entirely “black or white”

- If you know more about a specific topic or want to correct a statement, please do not hesitate!

Kahoot Quizzes

- Go to https://kahoot.it/
  - or scan QR code
- Enter game pin

Please introduce yourself

- Name
- Your day job
- Security Experience

1: Beginner 10: Pro

- What do you expect from this course?

Questions about whether design is necessary or affordable are quite beside the point: design is inevitable. The alternative to good design is bad design—not no design at all.

Douglas Martin

If you think good architecture is expensive, try bad architecture.

Brian Foote and Joseph Yoder

Security Criteria for Choosing a Language

Important criteria and examples

A Little Secret

- Some languages protect against certain vulnerability classes by design
- However, secure software *can* be written in *any* language
- But some just make it unnecessarily hard
- Mastering the language means mastering security

Security Criteria for Choosing a Language

But why is there so much low-quality code in specific languages?

- Some languages have very low entry barriers
- There will also be less skilled people writing and publishing code
- But that does not mean the language is bad

Security Criteria for Choosing a Language

- Memory safety
- Type safety
- (Parallelization support)
- Sandbox support
- Availability of secure frameworks

Memory Safety

Memory safety has many flavors

- Array bounds checks
- Pointer arithmetic
- Null pointers
- Accessibility of unallocated, de-allocated, or uninitialized memory

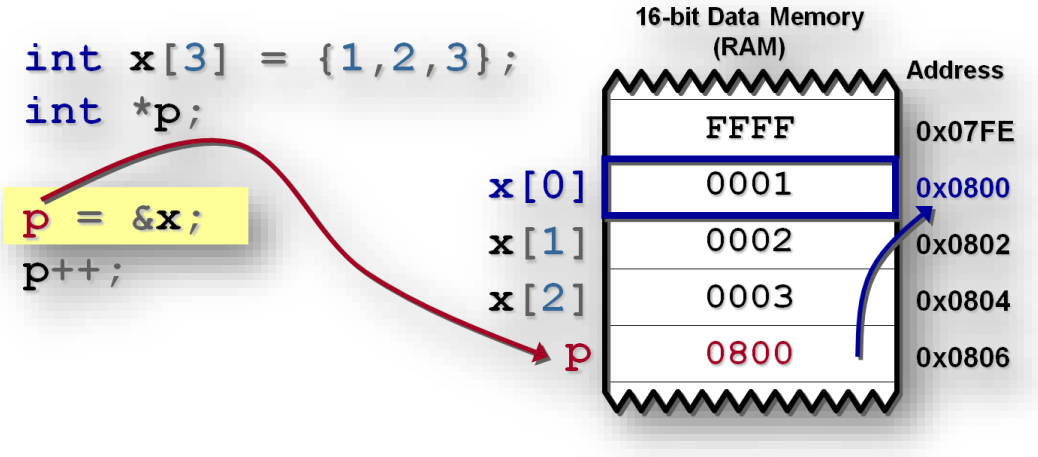


Image source: https://microchipdeveloper.com/tls2101:pointer-arithmetic

Memory Safety: Why Bother?

Non-memory-safe languages are susceptible to some vulnerability classes by design

- Buffer overflows
- Heap overflows
- Memory leaks
- Dangling pointers
- Format string vulnerabilities

Heartbleed

11 APR 2014 | NEWS

Open-source Bug Leaves Millions of Websites Exposed to Data Leaks

Heartbleed has potentially affected millions of websites

"The Heartbleed bug allows anyone on the internet to read the memory of systems protected by the vulnerable versions of the OpenSSL software", notes a report from Finnish security firm Codenomicon, who along with Google was one of the two companies who revealed the vulnerability earlier this week. As the company explained, in practice, the vulnerability allows encrypted communications over the internet to be compromised.

"This compromises the secret keys used to identify the services providers and to encrypt the traffic, the names

'/www.infosecurity-magazine.com/webinars/the-exploits-behind-the-headlines/

Why Not Watch?

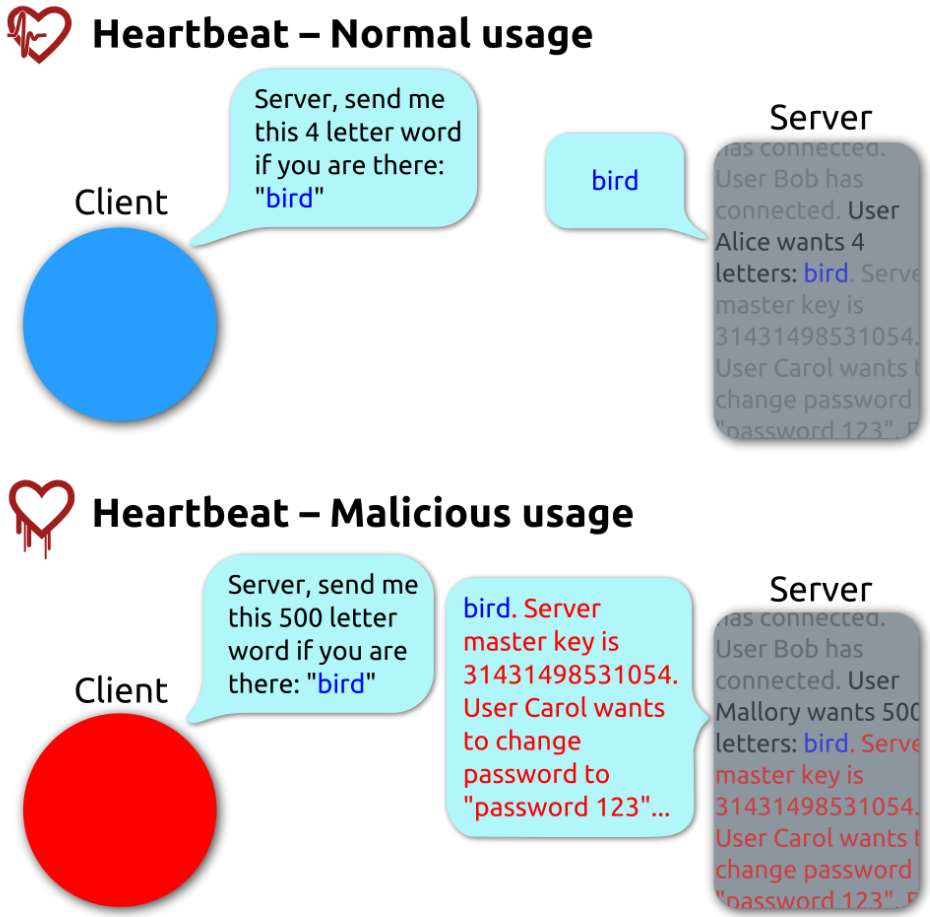
1 NOV 2012

Managing security – remotely: How to streamline your IT infrastructure

9 OCT 2014, 15: BST, 10:00 EDT

Shellshock: th Exploits behir the Headlines

Heartbleed: The Vulnerability



Memory Safety

Languages with no memory safety

- C
- C++
- Machine code

Languages with some form of memory safety

- Java
- C#
- Rust
- Go
- PHP
- Python
- …

Type System

Type strictness has two flavors

- Static vs. dynamic typing
- Strong vs. weak (loose) typing

Static vs. Dynamic Typing

Static typing: Types checked at compile time

Example: Java

```java
public int add(int a, int b) {
    return a + b;
}
System.out.println("The program is running!");
add(40, "2"); // Compile time error
```

Dynamic typing: Types checked during runtime

Example: Python

```python
def add(a: int, b: int) -> int:
    return a + b
print("The program is running!")
add(40, "2")
# Your IDE may warn you but the program
# throws an exception during runtime if
# ignored. Omit the type and your IDE won't
# warn you.
```

Strong vs. Weak (Loose) Typing

Strong typing: No automatic type coercion

Example: Python

```python
def add(a, b):
    return a + b
print("The program is running!")
add(40, "2")
# This will throw an error at runtime,
# because there is no automatic type
# coercion.
```

Weak typing: Automatic type coercion

Example: JavaScript

```javascript
function add(a, b) {
    return a + b;
}
console.log("The program is running!");
add(40, "2");
// You IDE will probably not warn you,
// and not even during runtime you'll
// see an error. The result is "402".
```

Loose Typing Example: JavaScript

```
> '5' - 3
2         // weak typing + implicit conversions * headaches
> '5' + 3
'53'      // Because we all love consistency
> '5' - '4'
1         // string - string * integer. What?
> '5' + + '5'
'55'
> 'foo' + + 'foo'
'fooNaN' // Marvelous.
> '5' + - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - + - - - '-2'
'52'      // Apparently it's ok

> var x * 3;
> '5' + x - x
50
> '5' - x + x
5       // Because fuck math
```

What About TypeScript?

How is TypeScript better than JavaScript?

- It adds static types to JavaScript
- It introduces a compilation step where types are checked
- However, type checks can be weakened (via configuration and via the use of any)
- It's still dynamic JavaScript at runtime (think dynamic assignment of JSON HTTP responses to variables)
- TypeScript is **statically** and **weakly** typed

Type System: Why Bother?

Why should we care?

- Stricter type systems have long-term advantages
- Less unexpected errors
- Faster refactoring
- Better IDE support (type hints)
- Better tool support (SAST)
- Sometimes even concrete vulnerabilities!

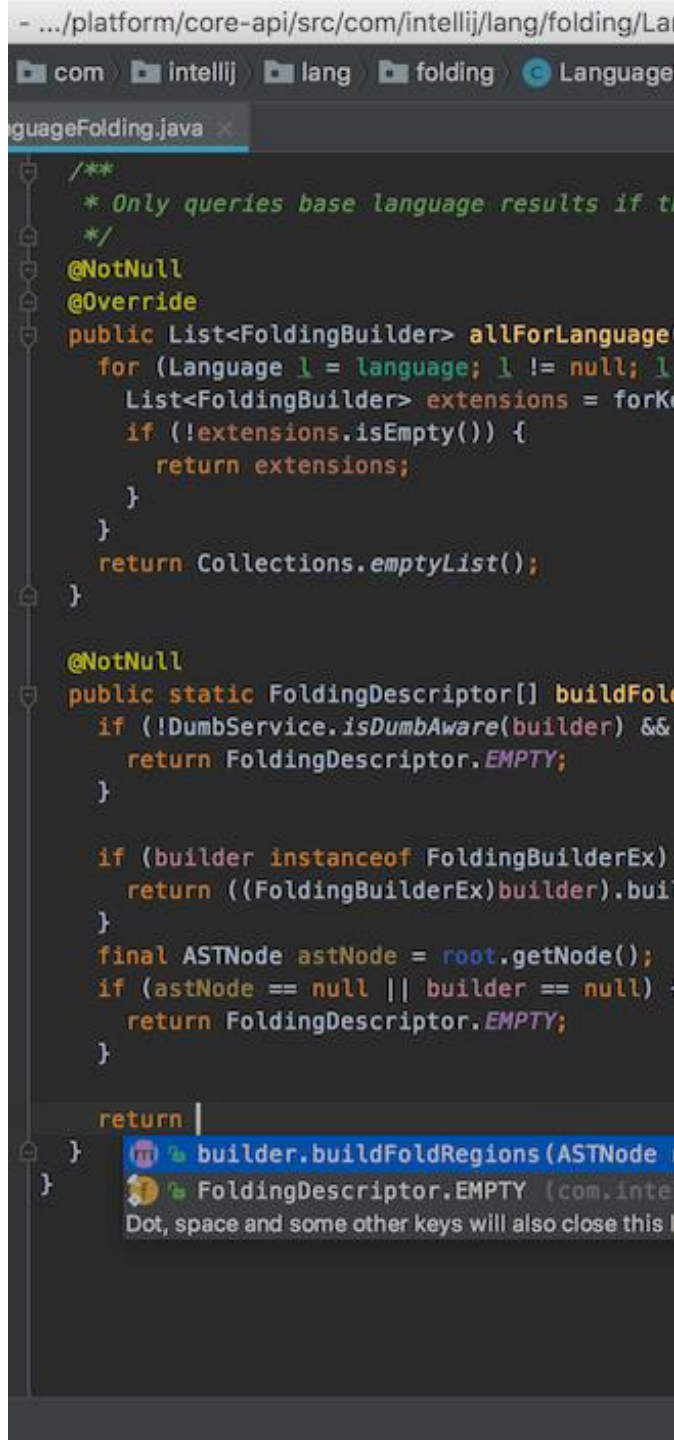


Image source: https://www.jetbrains.com/de-de/idea/

PHP Type Juggling Explained

- `'test' == 0`
  - Will return true
- `'test' == TRUE`
  - Will return true

Real-World Vulnerability

CMS Made Simple: Type Juggling

- One could manipulate request parameters so that `$tmp` was TRUE
- `true == 'a'` → TRUE
- Auth bypass (CVE-2018-10519)!

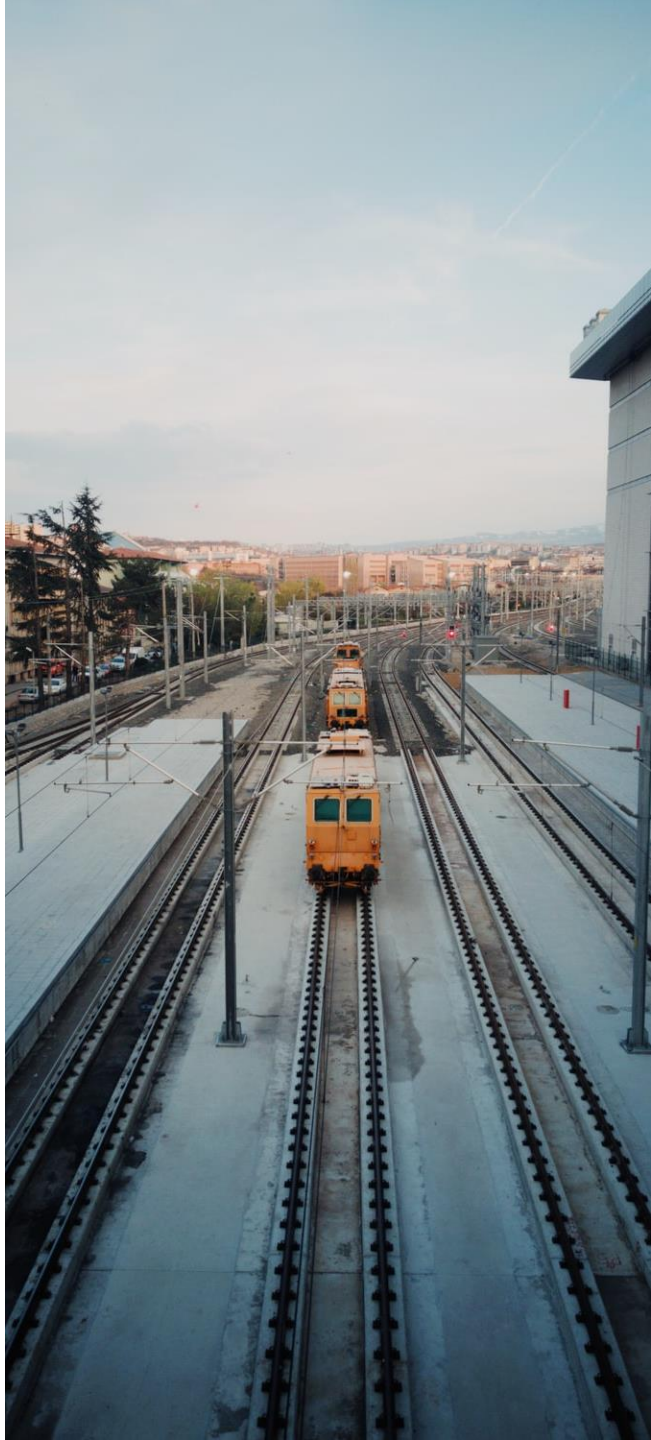
```php
protected function _check_passhash($uid,$checksum)
{
    $userops = \UserOperations::get_instance();
    $oneuser = $userops->LoadUserByID((int) $uid);
    if( !$oneuser ) return FALSE;

    $tmp = array(md5(__FILE__),$oneuser->password,$uid,\cms_utils::get_real_ip(),$_SERVER['HTTP_USER_AGENT']);
    $tmp = sha1(serialize($tmp));
    if ($oneuser && (string)$checksum != '' && $checksum == $tmp ) return TRUE;
    return FALSE;
}
```

Parallelization Support (Advanced)

This is an advanced topic!

- Some languages are designed for robust parallel computing (Clojure, Elixir, Erlang, Haskell, Rust, …)
- Others have less focus on parallelization

Inform yourself before you start!



Sandbox Support

Assume there will be vulnerabilities!

- Blast radius reduction is key to a sound security architecture
- Lock each process down to only the necessary capabilities
- Sandbox technology can help

Sandbox Support

Operating system level

- AppArmor
- SELinux
- seccomp
- Chroot
- Namespaces

Platform level

- Your web browser
- WebAssembly

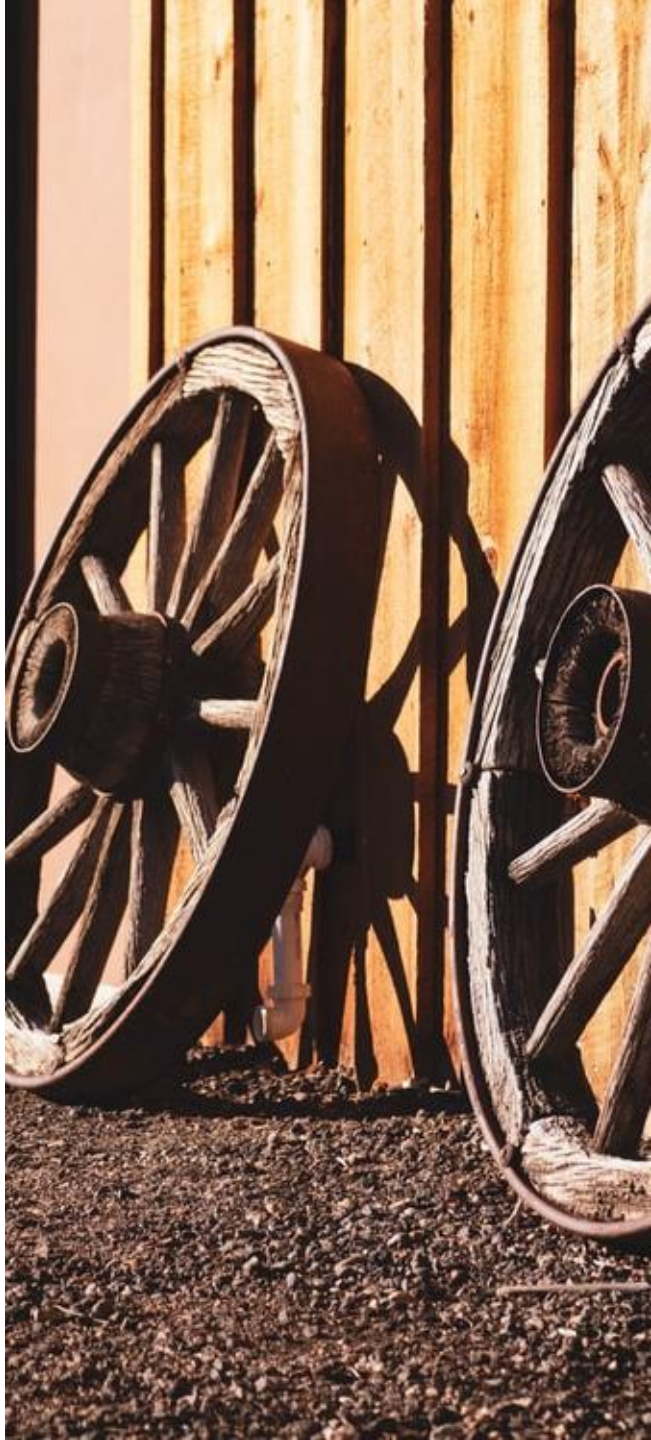
Language level

- .NET Code Access Security (CAS)
- Java Security Manager

Availability of Secure Frameworks

Do not reinvent the wheel!

- Build on proven technology if possible
- But only pull in what's strictly needed
- Framework availability might influence the language choice



Availability of Secure Frameworks

Typical jobs done by frameworks

- Authentication
- Session management
- Authorization
- Data persistence
- Templating
- Configuration
- …

Security Criteria for Choosing a Language

- Memory safety
- Type safety
- (Parallelization support)
- (Sandbox support)
- Availability of secure frameworks

Quiz!

Kahoot!

Security Criteria for Choosing a Language

Visit Kahoot and enter the game pin I'll share with you!

https://kahoot.it

Pitfalls in Low-level Languages

Are memory vulnerabilities still an issue nowadays?

2023 CWE Top 25 Most Dangerous Software Weaknesses

Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2022
1	CWE-787	Out-of-bounds Write	63.72	70	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.54	4	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	34.27	6	0
4	CWE-416	Use After Free	16.71	44	+3
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	15.65	23	+1
6	CWE-20	Improper Input Validation	15.50	35	-2
7	CWE-125	Out-of-bounds Read	14.60	2	-2
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.11	16	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.73	0	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	10.41	5	0
11	CWE-862	Missing Authorization	6.90	0	+5
12	CWE-476	NULL Pointer Dereference	6.59	0	-1
13	CWE-287	Improper Authentication	6.39	10	+1
14	CWE-190	Integer Overflow or Wraparound	5.89	4	-1
15	CWE-502	Deserialization of Untrusted Data	5.56	14	-3
16	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	4.95	4	+1
17	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.75	7	+2
18	CWE-798	Use of Hard-coded Credentials	4.57	2	-3
19	CWE-918	Server-Side Request Forgery (SSRF)	4.56	16	+2
20	CWE-306	Missing Authentication for Critical Function	3.78	8	-2
21	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.53	8	+1
22	CWE-269	Improper Privilege Management	3.31	5	+7
23	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.30	6	+2
24	CWE-863	Incorrect Authorization	3.16	0	+4
25	CWE-276	Incorrect Default Permissions	3.16	0	-5

https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html#top25list

2023 CWE Top 25 Most Dangerous Software Weaknesses

Rank	ID	Name	Score	CVEs in KEV	Rank Change vs. 2022
1	CWE-787	Out-of-bounds Write	63.72	70	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.54	4	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	34.27	6	0
4	CWE-416	Use After Free	16.71	44	+3
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	15.65	23	+1
					-2
					-2
					0
					0
					0
					+5
12	CWE-476	NULL Pointer Dereference	6.59	0	-1
13	CWE-287	Improper Authentication	6.39	10	+1
14	CWE-190	Integer Overflow or Wraparound	5.89	4	-1
15	CWE-502	Deserialization of Untrusted Data	5.56	14	-3
16	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	4.95	4	+1
17	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.75	7	+2
18	CWE-798	Use of Hard-coded Credentials	4.57	2	-3
19	CWE-918	Server-Side Request Forgery (SSRF)	4.56	16	+2
20	CWE-306	Missing Authentication for Critical Function	3.78	8	-2
21	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.53	8	+1
22	CWE-269	Improper Privilege Management	3.31	5	+7
23	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.30	6	+2
24	CWE-863	Incorrect Authorization	3.16	0	+4
25	CWE-276	Incorrect Default Permissions	3.16	0	-5

11 Web vulnerabilities
6 C/C++ only vulnerabilities (especially in the top categories)
8 General vulnerabilities affecting both Web and C/C++

https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html#top25list

Pitfalls in Low-level Languages

Potential vulnerabilities in non-memory-safe languages

- Buffer overflows
- Heap overflows
- Format string vulnerabilities

Buffer Overflows

Classical Buffer Overflow

```
void function foo(const char * arg)
{
    char buf[10];
    strcpy(buf, arg);
    […]
}
```

Buffer Overflow in C++

```
char buf[BUFSIZE];
cin >> (buf);
```

Buffer Overflows: The Problem

In RAM: Structured mix of data and code

- Program writes beyond memory area
- Overwriting control structures
- Modified behavior of the following program flow

Buffer Overflows: Countermeasures

What can we do about them?

- Don't write beyond the buffer, do bounds checks!
- Be careful with user input
- Use String and Vector classes in C++
- Do not use unsafe methods like `strcpy`
- C11/C18 Annex K: Bounds-Checking Interfaces

Stay in the "safe world" by using languages with automatic memory management

Format String

Format string: Threat

- Format string functions allow formatted input/output
  - E.g., `printf(), scanf()`
- Insecure code example:

```
int main(int argc, const char* argv[]) {
    if (argc < 2) { return 1; }
    printf("Hello, ");
    printf(argv[1]);
    return 0;
}
```

- Second `printf()` call allows inclusion of format string flags:
  - e.g.: Multiple statements of `%08x`: Reading the stack
  - `%s`: Read arbitrary memory location
  - `%n`: Write to arbitrary memory location

Format string: Threat

- Risk:
  - FS flags allow reading from and writing to an arbitrary memory location.
  - Attacker can run own code

- **Risky places**:
  - Usage of format string methods **and**
  - Format string comes from user

Format string: Counter measures

- Format string must never come from user
- The secure C functions like `sprintf_s()` do not allow the %n operator:
  - No write access for the attacker possible
- Secure code example:

```
int main(int argc, const char* argv[]) {
    if (argc < 2) { return 1; }
    printf("Hello, %s", argv[1]);
    return 0;
}
```

Quiz!

Kahoot!

Pitfalls in Low-level Languages

Visit Kahoot and enter the game pin I'll share with you!

https://kahoot.it

Input Handling

Input Handling

- **Input handling has three major activities**
  - Canonicalization
  - Input validation
  - Sanitization

Canonicalization

Question: Are someone@example.com and Someone@example.com the same email address?

Canonicalization

Question: Are someone@example.com and Someone@example.com the same email address?

The answer is a clear **yes and no**.

Canonicalization: Why Should We Care?

- **In the case of email**
  - The uniqueness is often important for security
  - Not canonicalizing it might make impersonation possible
- **Other examples**
  - IP addresses (127.0.0.1 vs. 2130706433)
  - URLs (https://www.a.com vs. https://www.a.com/ vs https://www.a.com:443/ vs. …)

Input Validation

Checking inputs against certain formats

- Maximum length
- Allowed characters
- Date and time
- Boolean values
- Email addresses
- …

Input Validation

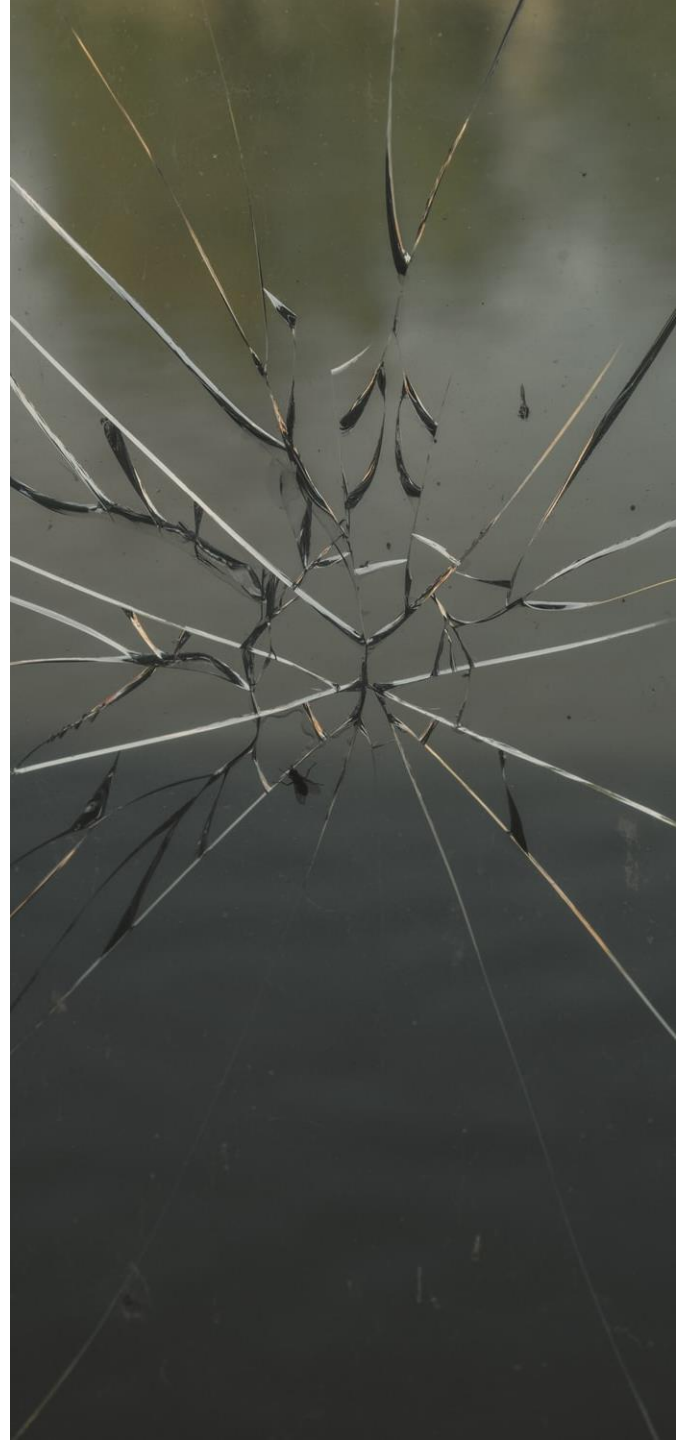
Allowlist (whitelist) vs Blocklist (blacklist)

- Instead of disallowing certain dangerous characters (**blocklist**) – define which characters should be safely allowed (**allowlist**)
- Blocklists are always in danger to miss a certain dangerous character

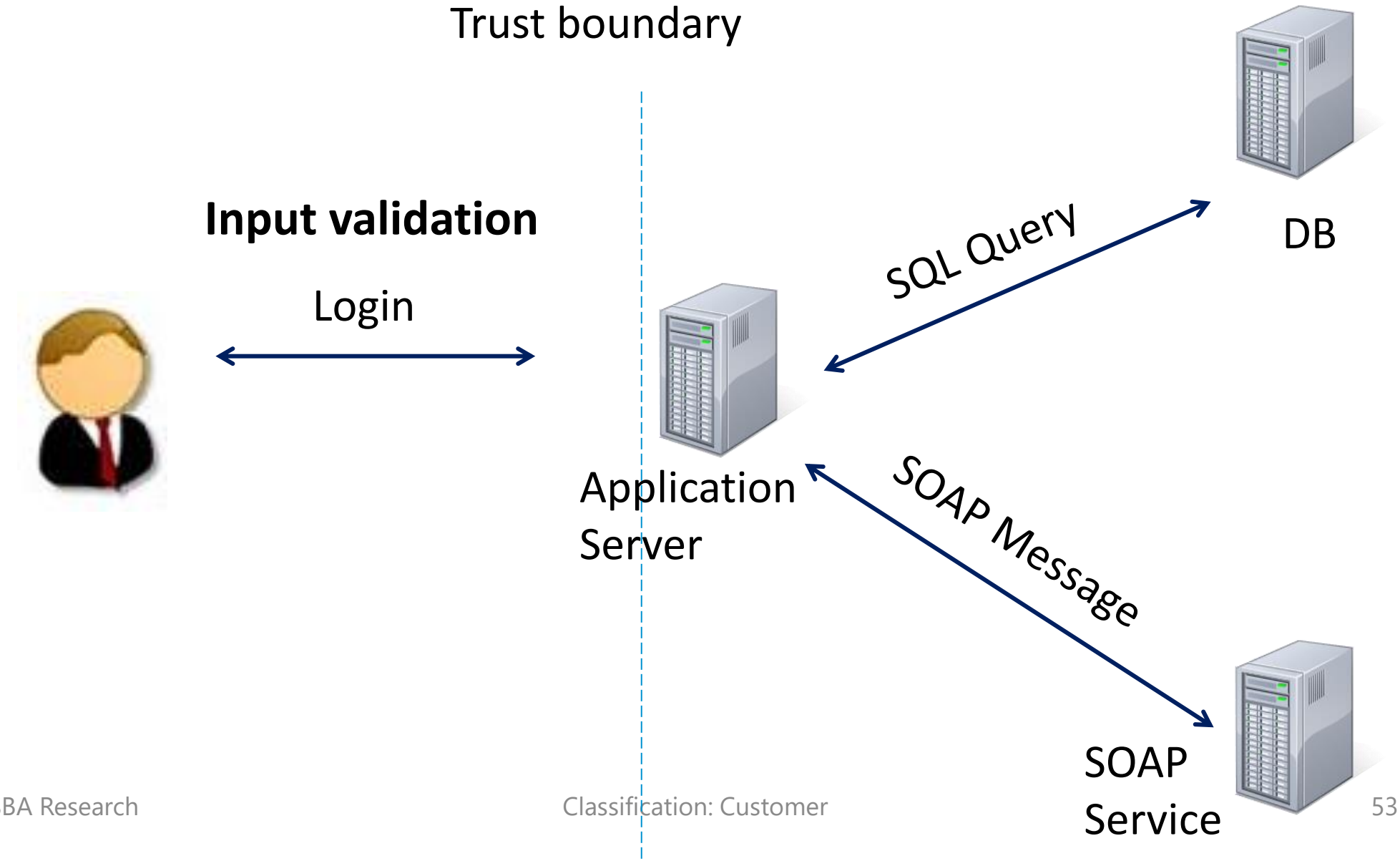
Input Validation

Does input validation help against specific vulnerabilities?

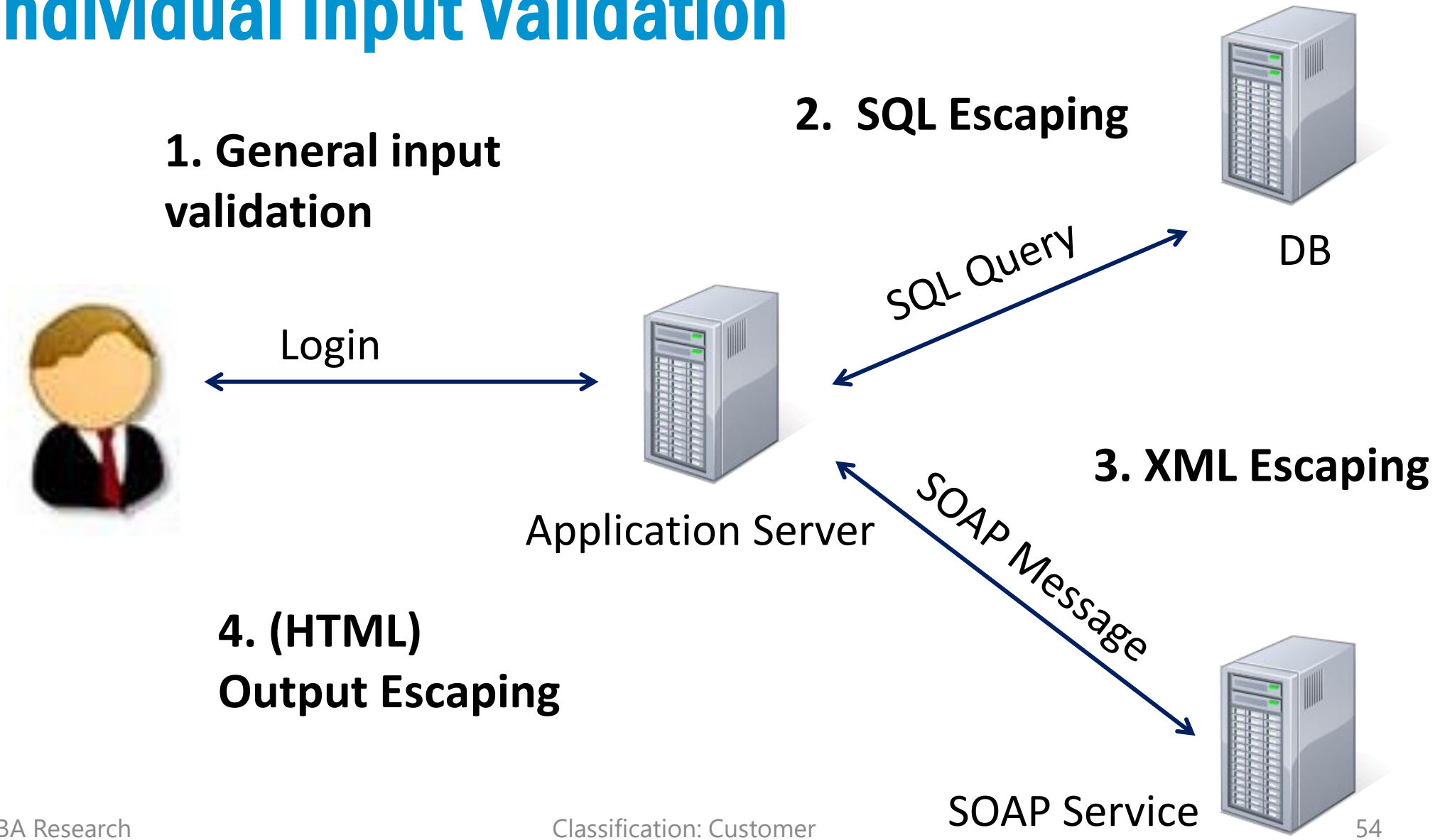
- Usually, no!
- But it’s a good generic measure to reduce the attack surface
- Think SQL injection vs. allowing the ' character
- Output encoding is the key!



Input Validation at the Border?



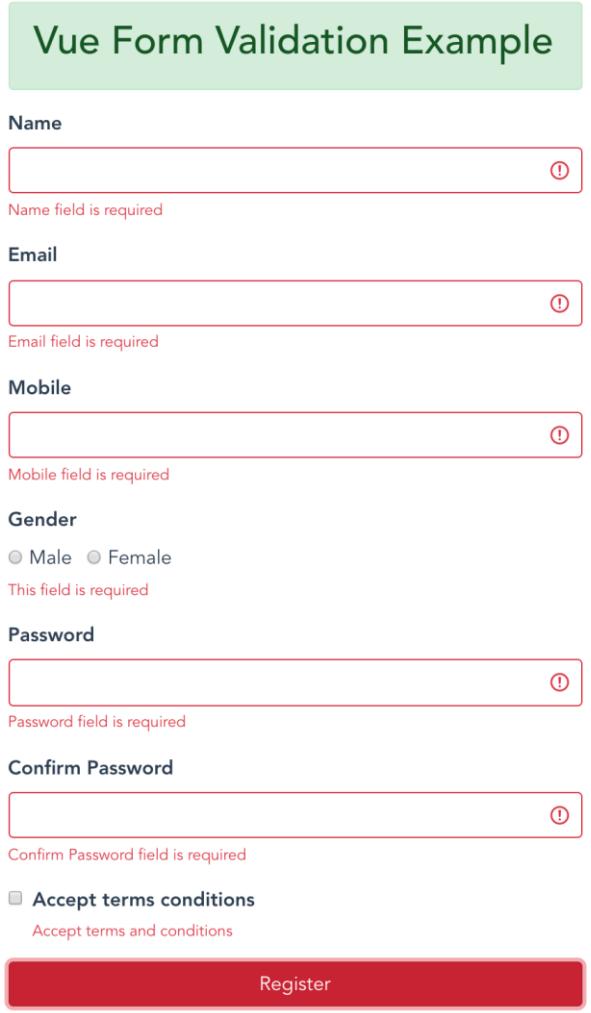
Individual Input Validation



Input Validation on the Client Side?

Is there something wrong with client-side validation?

- No, if there is a server-side counterpart
- The web app will be faster as you save a server round trip per form submission
- Good for usability
- Have both in place!



Demo time 

Damn Vulnerable Web Application: File Upload

Output Encoding

What is it?

- Safely embedding user input into different data structures
- Converting characters that have a special meaning in the target syntax
- Avoiding the possibility to change the parent data structure

Image source: https://stackoverflow.com/questions/54343557/how-to-display-encoded-html-in-browser

Output Encoding

What are common situations where I must be aware of the dangers, and encode inputs?

- HTML
- JavaScript
- XML
- CSV
- LDAP
- SMTP
- ~~SQL~~ – special case, use parameterization
- ~~Shell~~ – special case, avoid if possible
- …

Output Encoding: Special Cases

SQL Injection

- There are some pitfalls with encoding
- Use prepared statements / parameterized queries

Command Injection

- There is no safe way to encode inputs
- There are too many contexts to watch out for
- Just don’t do it!
- Use a very strict allowlist if not possible otherwise

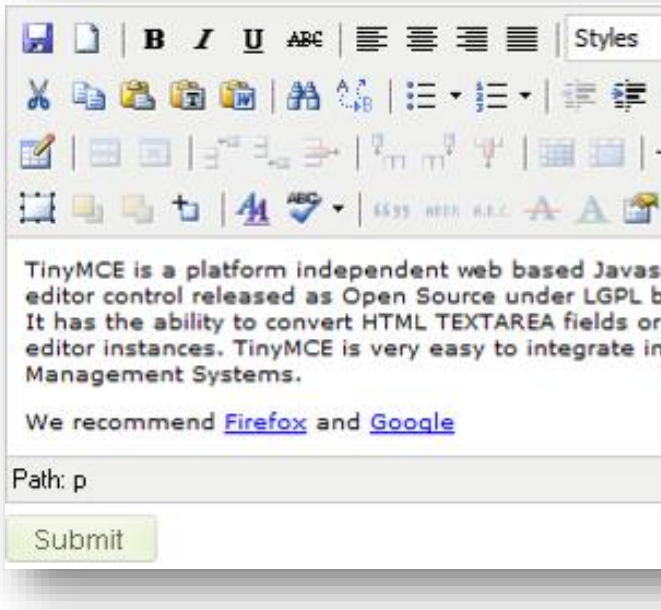
Output Encoding

As opposed to input validation, this is usually the primary protection mechanism!

- Input validation lowers the attack surface
- Output encoding protects against specific vulnerability classes

Sanitization

- There will be situations where you **must output HTML directly**
- Think text that can be formatted (WYSIWYG)
- In this case, we must get rid of the "dangerous" parts, e.g., everything that may contain JavaScript
- This is called **sanitization**!

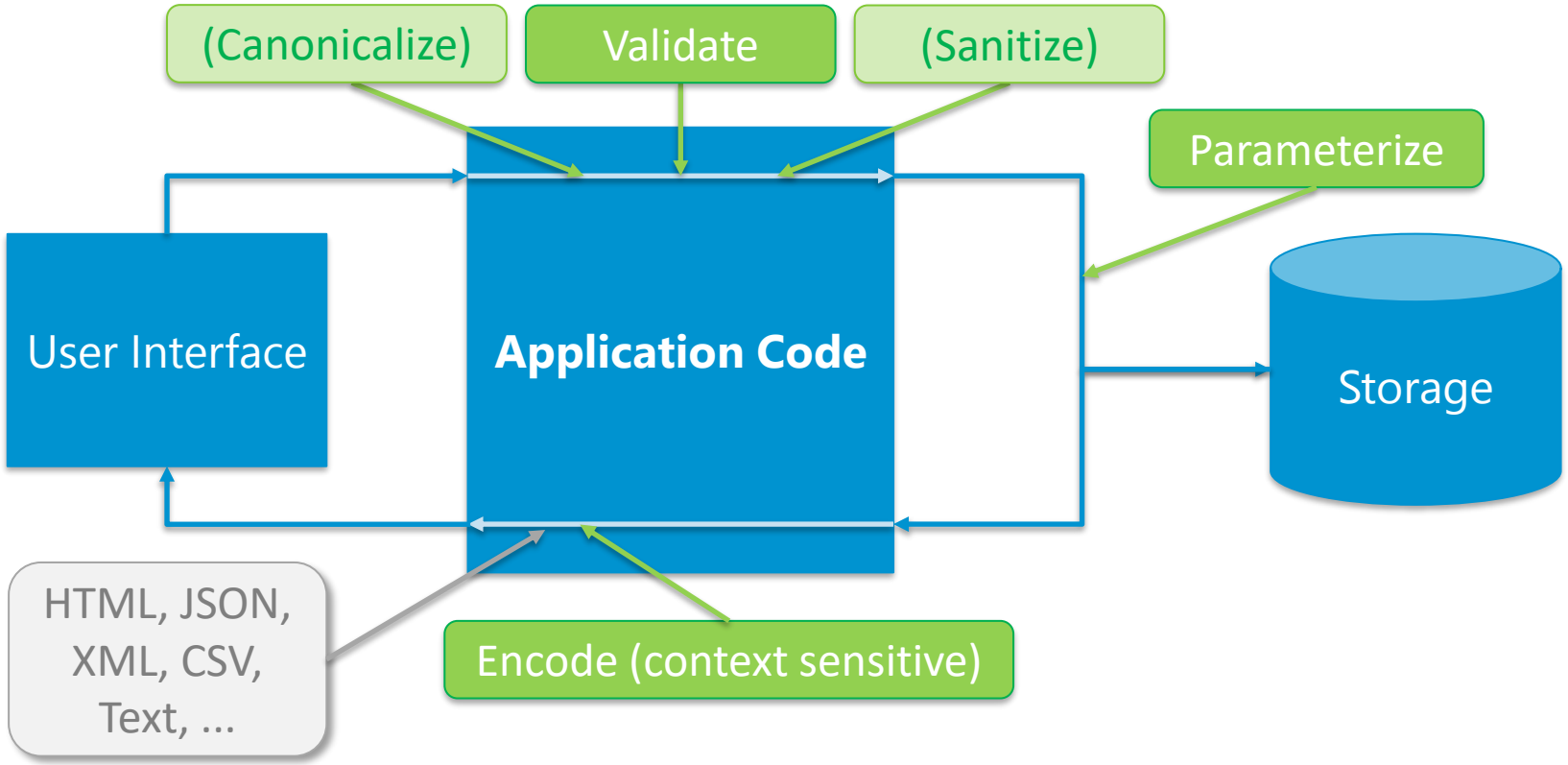


Sanitization: The Rules

Don’t roll your own sanitizer!

- There are specialized libraries!
- DOMPurify (JavaScript)
- Angular comes with its own
- HTML Purifier (PHP)
- OWASP Java HTML Sanitizer
- HtmlSanitizer (C#)
- …

(Canonicalize), Validate, (Sanitize), Parameterize, Store, Encode



Recommendation: Template Engine

- Use a template engine that escapes all output by default
- No more manual HTML escaping necessary
- Be aware of the context you are outputting

Output escaping: It's about context

```
<html>
    <head>
        <script>
            var a = '{{ output }}';
            var b = "{{ output }}";
            var {{ output }} = 'value'; // {{ output }}
        </script>
    </head>
    <body>
        <h1>{{ output }}</h1>
        <p attr="{{ output }}" attr='{{ output }}' attr={{ output }}
            {{ output }}="value" style="color: #{{ output }}">
            {{ output }}
        <{{ output }}></{{ output }}>
        </p>
    </body>
</html>
```

Demo time 

Damn Vulnerable Web Application: Cross-Site Scripting

Quiz!

Kahoot!

Secure Coding Practices

Visit Kahoot and enter the game pin I'll share with you!

https://kahoot.it

Identification, Authentication, and Authorization

Identification

- Assertion of a unique identity
- Critical first step in applying access controls

Authentication

- Verifying identity of a user
- Combination of identity and information
- Something you have, know or are

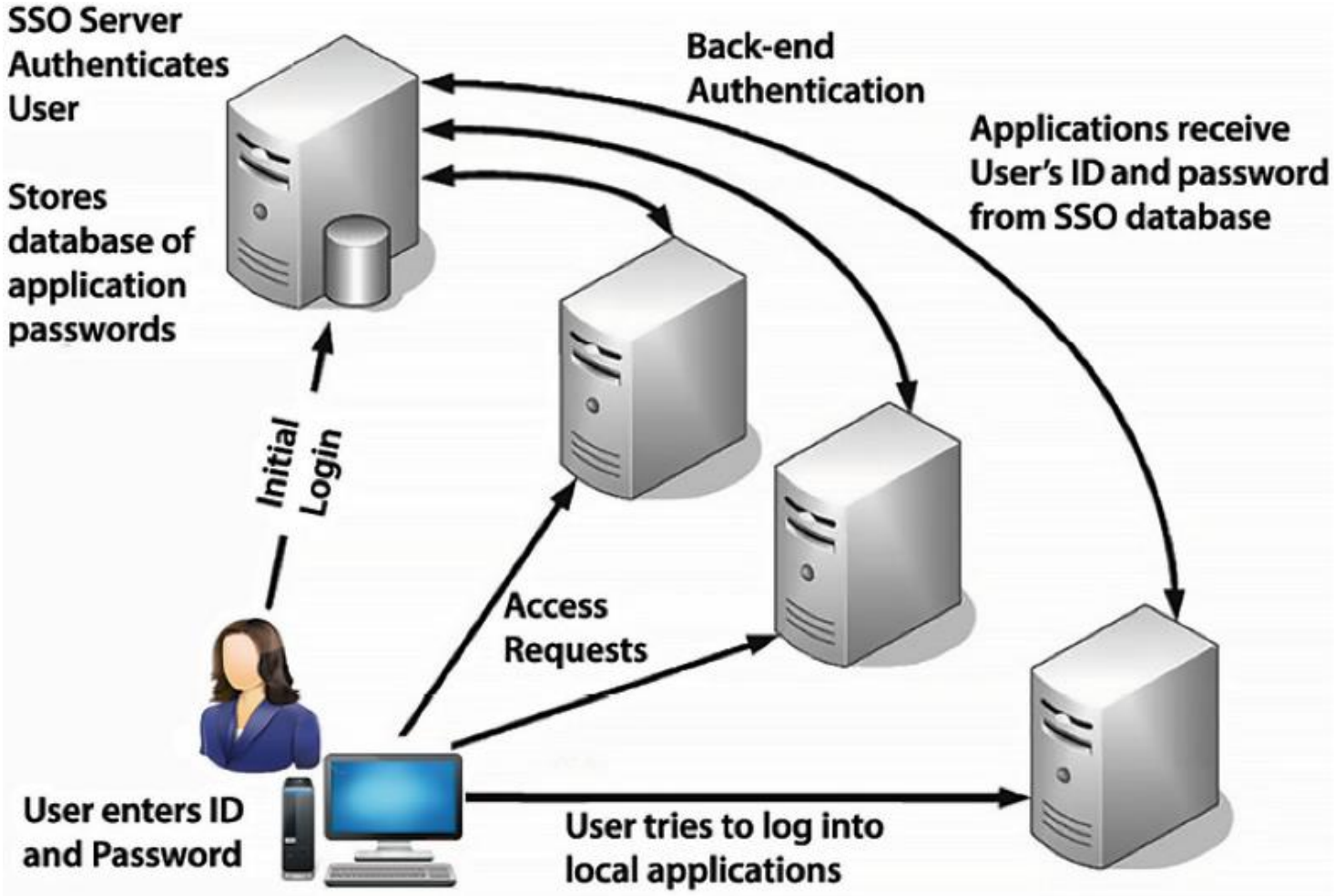
Authorization

- Determining type of access

Authentication: Attack Vectors

- Weak passwords
- Password reset process
- Brute-force attacks
- No multi-factor authentication
- Attacks go unnoticed

Single Sign-On



Source: Adam Gordon, Official (ISC)² Guide to the CISSP CBK, Fourth Edition

Single Sign-On Advantages

- Efficient log-on process
- No need for multiple passwords
- Users may create stronger passwords
- Standards can be enforced across entire SSO system
- Centralized Administration

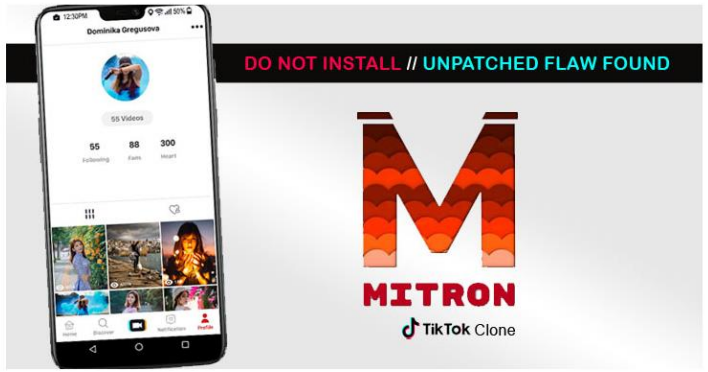
Single Sign-On Disadvantages

- Additional costs to ensure availability
- All credentials of a user are protected by a single password
- Enterprise level SSO architecture is complex and requires significant integration

Broken Authentication in the Real World

Exclusive – Any Mitron (Viral TikTok Clone) Profile Can Be Hacked in Seconds

May 30, 2020 Mohit Kumar



Popular This Week


Microsoft Releases Urgent Windows Update to Patch Two Critical Flaws


e-Commerce Site Hackers Now Hiding Credit Card Stealer Inside Image Metadata


Russian Hacker Gets 9-Year Jail for Running Online Shop of Stolen Credit Cards


'Satori' IoT DDoS Botnet Operator Sentenced to 13 Months in Prison

The security issue discovered by Indian vulnerability researcher Rahul Kankrale resides in the way app implemented 'Login with Google' feature, which asks users' permission to access their profile information via Google account while signing up but, ironically, doesn't use it or create any secret tokens for authentication.

In other words, one can log into any targeted Mitron user profile just by knowing his or her unique user ID, which is a piece of public information available in the page source, and without entering any password—as shown in a video demonstration Rahul shared with The Hacker News.

https://thehackernews.com/2020/05/titok-mitron-app-hacking.html

Password Guidelines

800-63-3: Digital Identity Guidelines

- https://pages.nist.gov/800-63-3/sp800-63b.html
- Major overhaul in June 2017
- Major changes
  - Remove periodic password change requirements
  - Drop the password complexity rules!
    - No more `Password1!`
  - Require screening of new passwords against lists of commonly used or compromised passwords
  - Requires two factor authentication for higher confidence levels

800-63-4: Digital Identity Guidelines

- https://pages.nist.gov/800-63-4/
- July 2025: Release of final SP 800-63, Revision 4
  - Last major overhaul before that happened in June 2017
- Summary
  - The overall philosophy is the same: balance security and usability, avoid forcing users into bad patterns.
    - But the 2025 revision tightens rules, making some previously recommended practices mandatory.
  - In systems relying solely on passwords, the minimum length requirement is significantly increased (to 15).
  - Prohibition of composition rules is stronger.
  - Use of blocklists (banned/compromised password screening) is now more strictly required.
  - The encouragement of strong / phishing-resistant authentication methods (beyond just password + OTP) is more forceful.
  - Usability features are more firmly supported in the standard (e.g. showing typed characters briefly, typo tolerance).

Secure Password Hashes

- Designed to be slow and hard to compute, even for specialized hardware
- Bcrypt (1999)
  - Widespread support, still considered secure, some pitfalls exist (e.g., truncates after 72 characters, input truncated at null bytes)
- Argon2 (2015)
  - Uses modern cryptography and provides more protection against offline attacks, but is currently not available for every language and framework

How to Check If a Password Has Been Leaked 1/2

- Pwned Passwords API by Troy Hunt
- List of over 500 million compromised passwords
  - Upload first five chars of SHA1 hash of the password (E.g., https://api.pwnedpasswords.com/range/5BAA6)
  - Service returns all known hashes which start with these five characters including how often they have been leaked
- Example:

How to Check If a Password Has Been Leaked 2/2

- Check if the entire hash is present in the response
- On average more than 500 hashes are returned
  - The Pwned Passwords server cannot know exactly which password was asked for
  - k-anonymity around 500
- Offline alternative
  - Download the Pwned Passwords list
  - Can be downloaded as list of SHA-1 or NTLM-hashes

NIST Guideline vs. Microsoft Windows

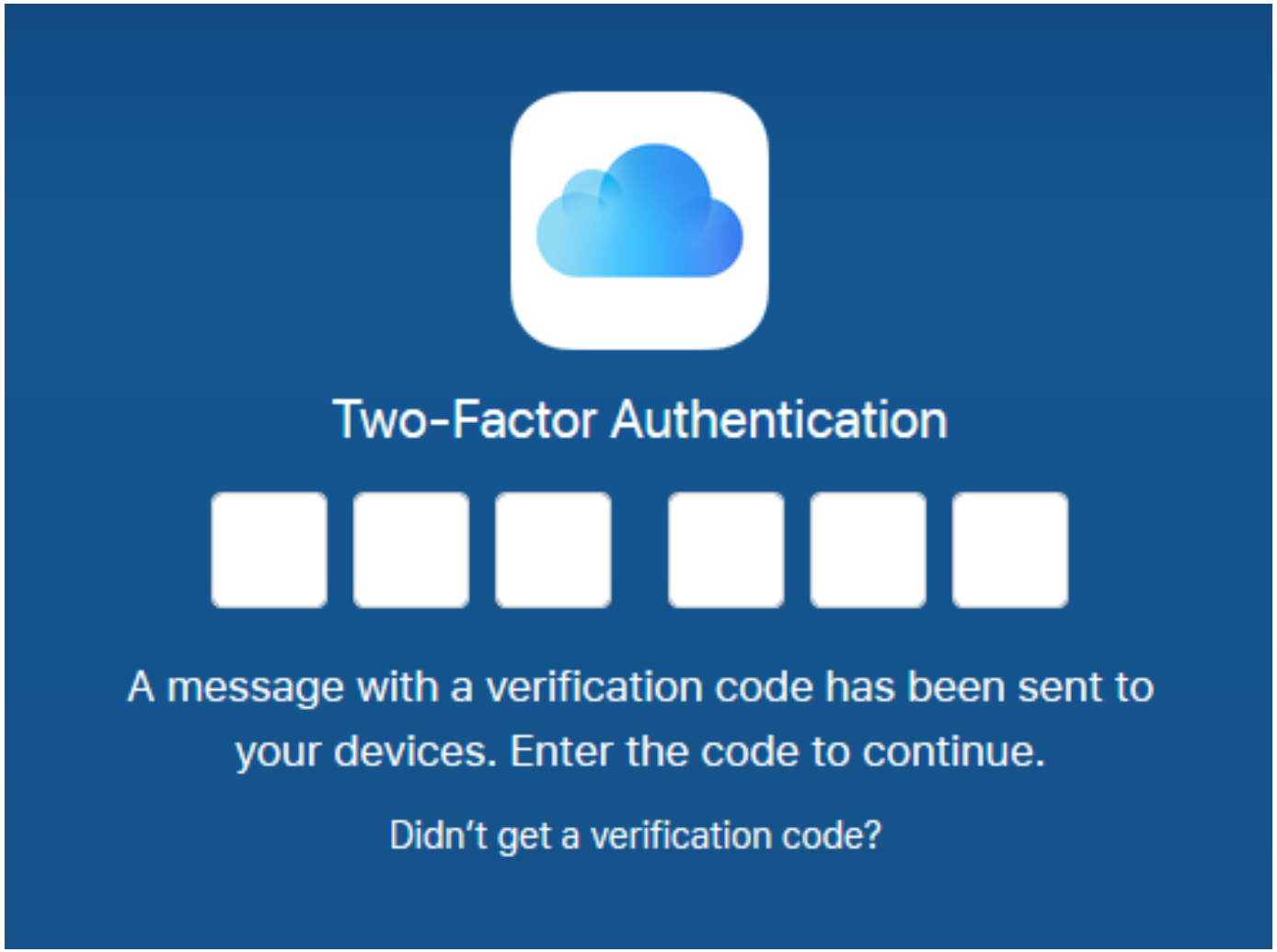
- **Question**: Is the NIST guideline is suitable for Windows AD accounts?

Why the NIST Guideline Is Not Suitable for Windows AD Accounts

- NT LAN Manager (NTLM) hashes are used in a Windows environment
- This hash algorithm doesn't provide enough protection against offline brute force attacks
- Thus, the prerequisites for NIST Assurance Level 1 are not fulfilled
  - We cannot simply relax the password complexity rules

Multi-Factor Authentication

MFA Example: iCloud Login

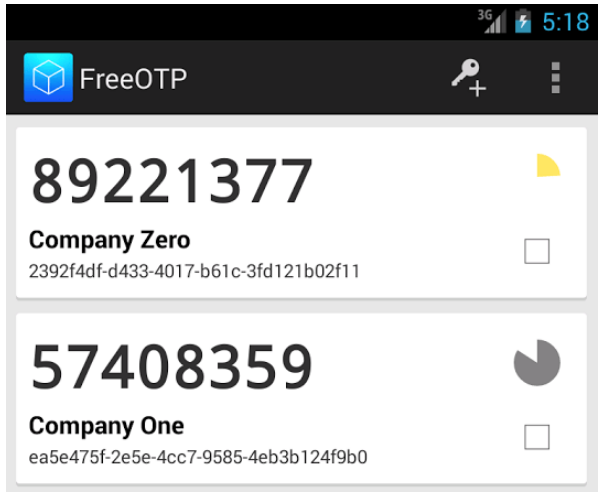


Authentication Factors

- Something You **Know** → Password or PIN
- Something You **Have** → Token or smart card
- Something You **Are** → Biometrics, such as a fingerprint
- **Single-factor** authentication → One of the factors
- **Multi-factor** authentication (MFA) → Combination of at least two factors

“Something You Have”: Form Factors

- **Apps** for **smartphones… (soft token implementation)**
  - E.g., FreeOTP, Google Authenticator



- **Tokens (hardware token implementation)**
  - Contain a secret based on asymmetric or symmetric keys

One-Time Password (OTP)

- A password that is valid for only one login or a single transaction
  - != static password
- One-time passwords != One time pad
  - One-time pad is an unbreakable encryption technique
- Example
  - For authorizing a banking transaction, a user must enter a TAN that is sent to the mobile device via text message
  - A hardware or software tokens generates one-time passwords (= something a user has)

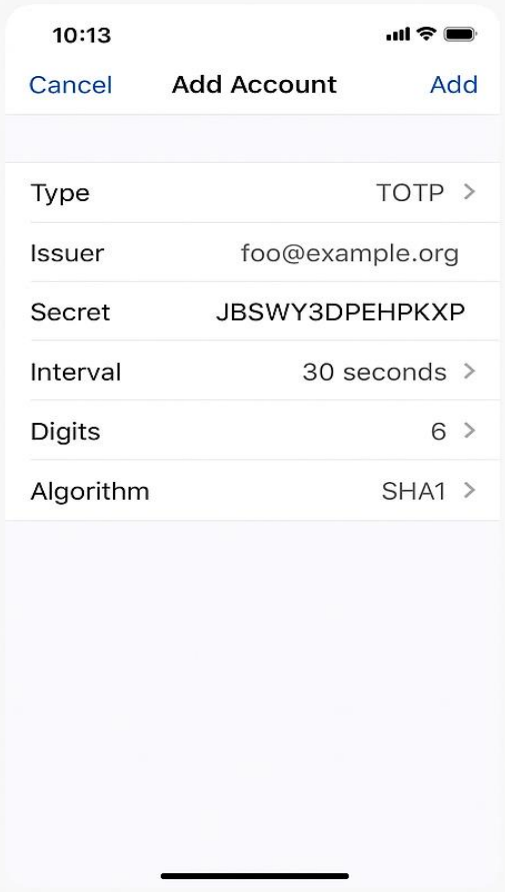
HMAC-based One-time Password Algorithm (HOTP)

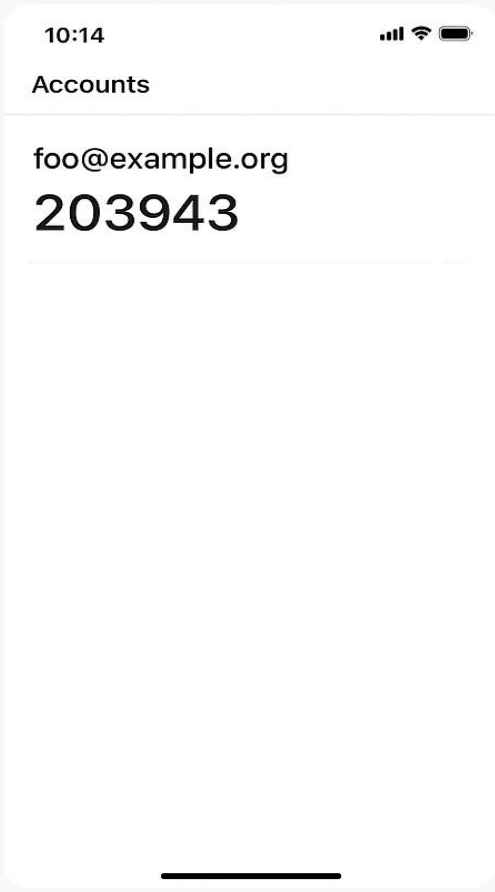
- RFC 4226, December 2005
- One of the first attempts of standardization
- A mathematical **algorithm** to generate a new password **based on the previous password**
  - OTPs are effectively a chain and must be used in a predefined order
- **HOTP**(*K*,*C*) = *Truncate*(*HMAC*(*K*,*C*)) & 0x7FFFFFFF
  - K: secret key
  - C: counter
  - HMAC: keyed-hash message authentication code
- A HOTP value is a 6–8-digit number

Time-based One-Time Password Algorithm (TOTP)

- RFC 6238
- Computes a one-time password from a shared secret key and the current time
- TOTP = HOTP(SecretKey, TC),
  - TC: Timestamp Counter (usually 30s granularity)
- Weaknesses:
  - TOTP codes can be phished but must be used by the attacker in real time
  - Brute-Forcing of codes possible if number of login attempts is not limited
  - If an attacker manages to steal the shared secret new TOTP codes can be generated at will

Configuring TOTP



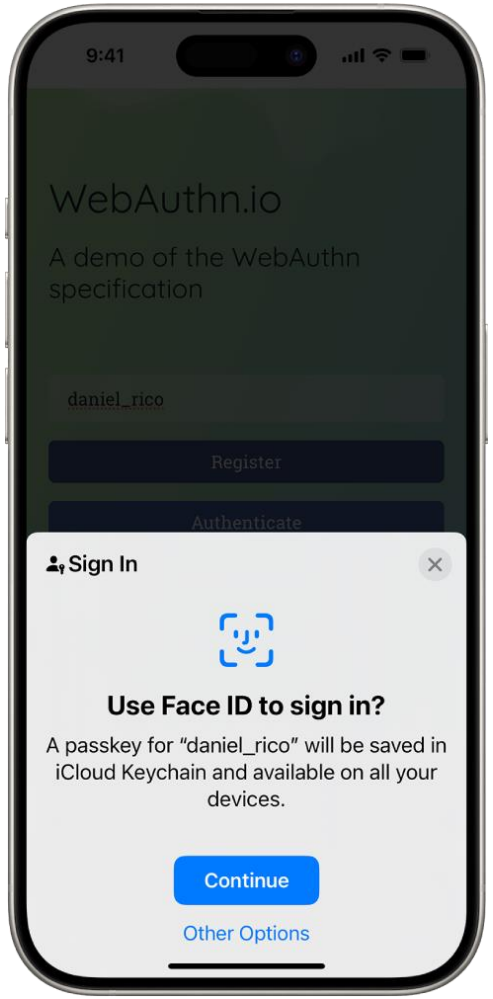


FIDO2 and WebAuthn

- Open authentication standard for 2-FA
  - Authentication with a simple touch
  - No driver installation necessary (HID device)
- Successor of FIDO U2F (Universal Second Factor)
- Initially developed by Google und Yubico
- Twitter, Facebook, Gmail, GitHub, Dropbox, and hundreds more compatible services
- Support in all major web browsers
- Protects against phishing

Passkeys

- User-facing implementation of WebAuthn in password managers
- Usually integrate some kind of cloud synchronization
- UX depends on the platform
- No easy transfer between platforms



Real-world Example 1

- Spot the mistake:

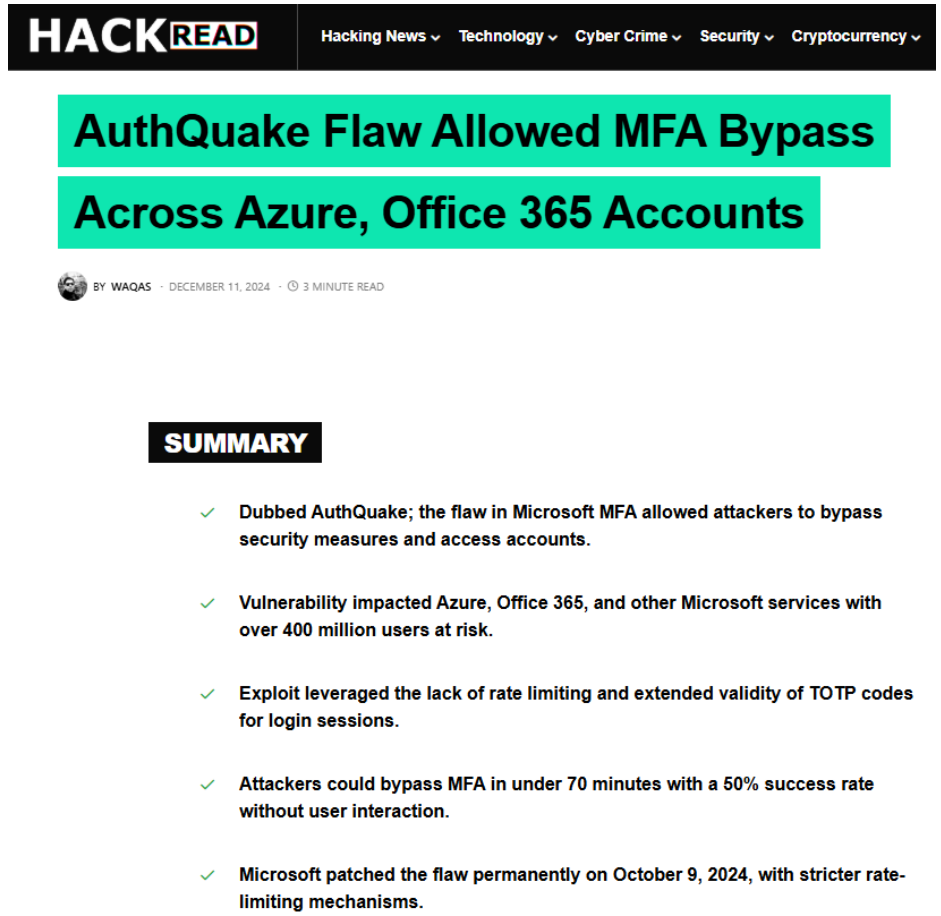
```
3 references
public static bool TFAVerify(ICredentials credentials, string code)
{
    try
    {
        var secret = [redacted]Util.Strings.StringToSHA256(credentials.Login);
        Google.Authenticator.TwoFactorAuthenticator ga = new Google.Authenticator.TwoFactorAuthenticator();
        return ga.ValidateTwoFactorPIN(secret, code);
    }
    catch
    {
        return false;
    }
}
```

Real-world Example 1: Insecure TOTP Secret

```
3 references
public static bool TFAVerify(ICredentials credentials, string code)
{
    try
    {
        var secret = [redacted]Util.Strings.StringToSHA256(credentials.Login);
        Google.Authenticator.TwoFactorAuthenticator ga = new Google.Authenticator.TwoFactorAuthenticator();
        return ga.ValidateTwoFactorPIN(secret, code);
    }
    catch
    {
        return false;
    }
}
```

- Secret depends on the username!
- Secret should be randomly generated!

Real-world Example 2: Missing Rate Limiting



HACKREAD | Hacking News | Technology | Cyber Crime | Security | Cryptocurrency

AuthQuake Flaw Allowed MFA Bypass Across Azure, Office 365 Accounts

BY WAQAS · DECEMBER 11, 2024 · 3 MINUTE READ

SUMMARY

- Dubbed AuthQuake; the flaw in Microsoft MFA allowed attackers to bypass security measures and access accounts.
- Vulnerability impacted Azure, Office 365, and other Microsoft services with over 400 million users at risk.
- Exploit leveraged the lack of rate limiting and extended validity of TOTP codes for login sessions.
- Attackers could bypass MFA in under 70 minutes with a 50% success rate without user interaction.
- Microsoft patched the flaw permanently on October 9, 2024, with stricter rate-limiting mechanisms.

Insecure Generation of QR Code

- TOTP secret is transmitted insecurely!
- Three problems exist:
  - Secret is transmitted
    - To a 3rd Party (e.g., Google)
    - As a URL parameter
    - Per HTTP instead of HTTPS

MFA Bypasses

- Cisco Talos reports that in the first quarter of 2024, 50% of their incident responses involved MFA bypass attacks.
- **MFA fatigue**
  - Uber Breach in Sept 2022 by Lapsus$ hacking group, repeatedly send MFA requests to a contractor
- **Session Hijacking**
- **Man-in-the-middle**
- **Social Engineering**
- **SIM Swapping**

How to prevent and mitigate MFA bypasses

- Implement biometric authentication
- Institute strong password policies
- Restrict login attempts
- Use phishing-resistant MFA, such as FIDO2 authentication
- Implement fraud detection controls

Quiz!

Kahoot!

Identification and Authentication Failures

Visit Kahoot and enter the game pin I'll share with you!

https://kahoot.it

Applied Cryptography

Cryptography

Primitives

- Block ciphers (AES)
- Stream ciphers (ChaCha20)
- Hash functions
- Public key primitives (Factoring, Elliptic Curves)

Schemes

- Symmetric crypto systems
- Asymmetric crypto systems
- Message authentication code (MAC)

Protocols

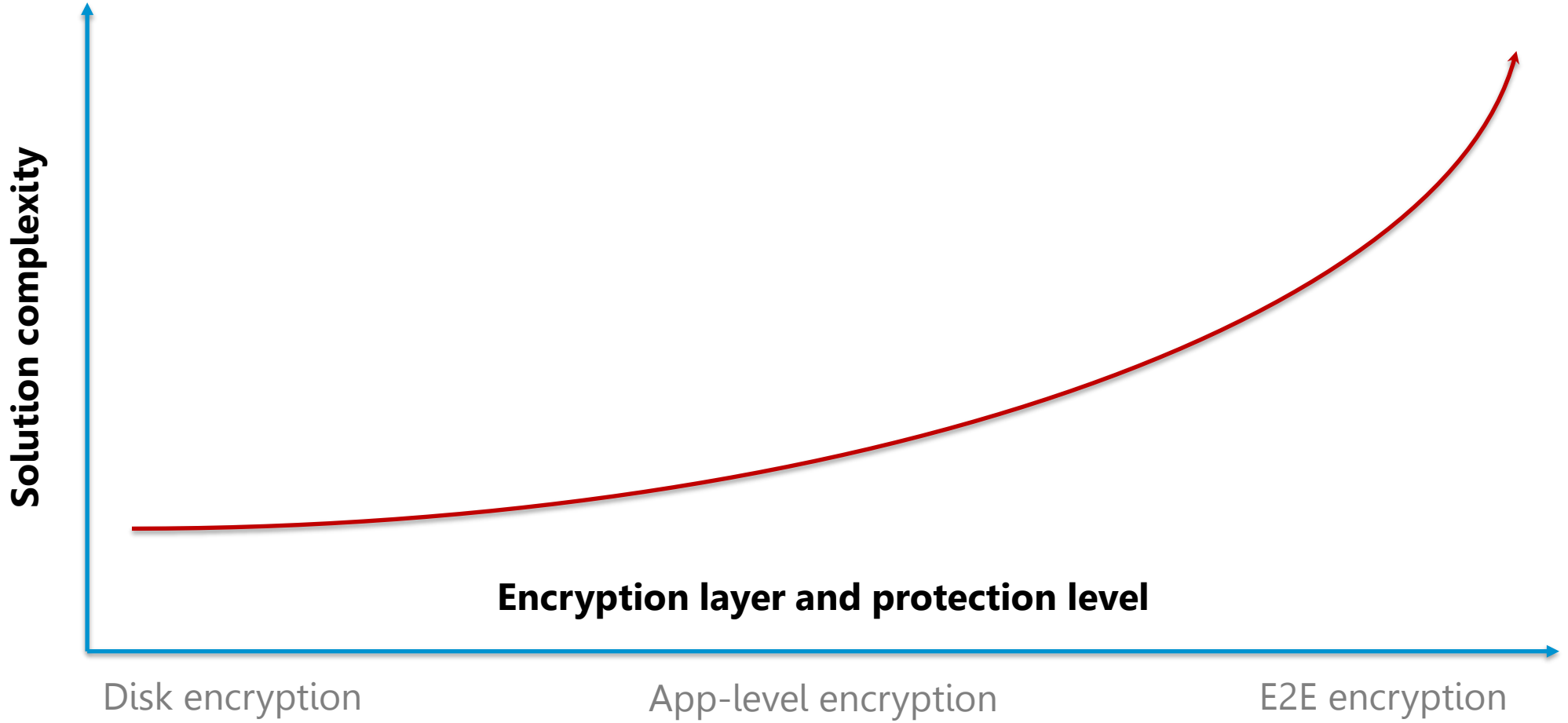
- TLS
- SSH
- IPSec
- S/MIME

Cryptography

Algorithm overview

- Secure random number generators
- Hash algorithms (SHA256, Blake2, …)
- Key derivation functions (Argon2id, PBKDF2, …)
- Key exchange algorithms (Diffie-Hellman, ECDHE, X25519, …)
- Symmetric encryption algorithms (AES-GCM, ChaCha20-Poly1305, …)
- Asymmetric encryption algorithms (RSA, ECIES, …)
- Message authentication codes (HMAC, Poly1305, …)
- Digital Signatures (RSA, EdDSA, ECDSA, …)

Cryptography: End-to-end Encrypt It All?



End-to-End Encryption: Things to Consider

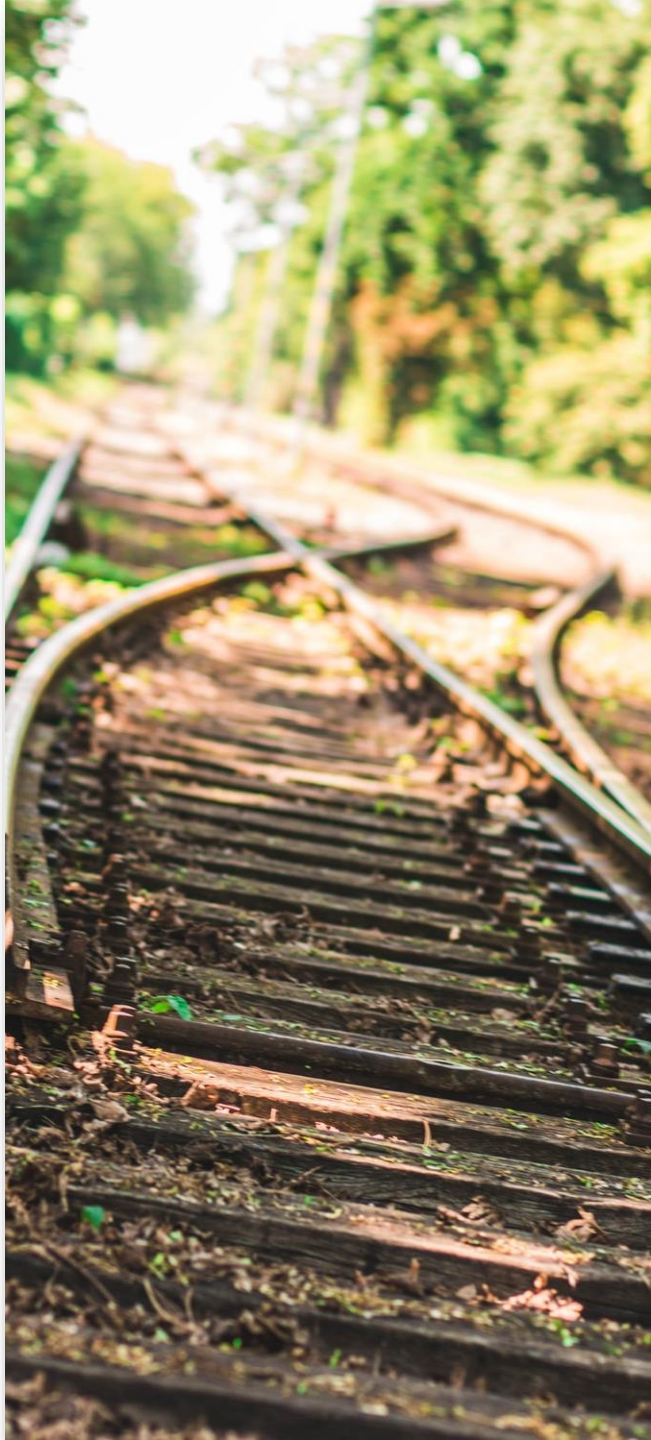
All these are hard to do

- Key recovery
- Backup
- Multi-device
- Database indexing
- Search
- Scalability
- …

Crypto Isn't Usually Broken

Cryptography is usually not broken but bypassed

- Applying cryptography is more about flaws than bugs.
- Ridiculous key lengths add nothing to security.
- Key distribution and management are the most difficult problems.



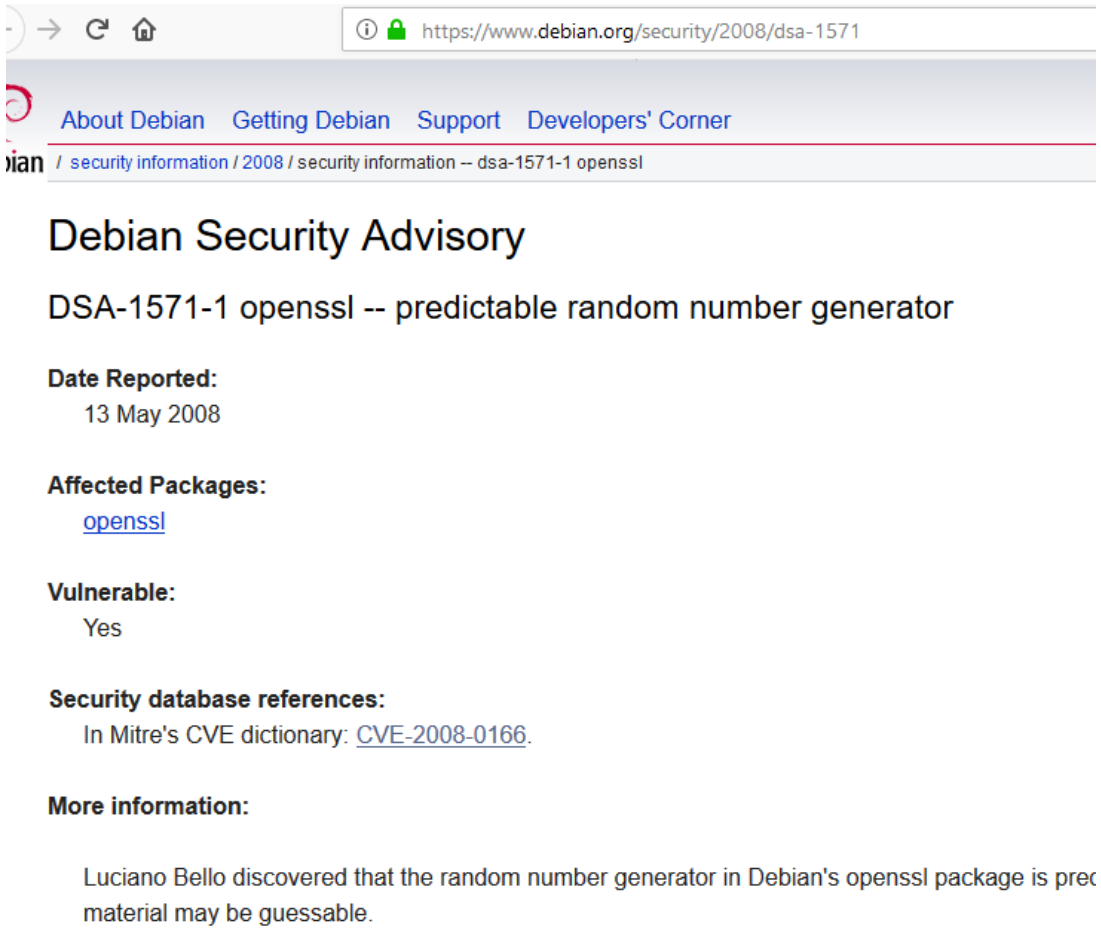
Random Numbers

4, 4, 4, 4, 4, ...

Why Do We Need Random Numbers?

- Keys for symmetric encryption
  - Key must be kept secret
  - **Key must be selected uniformly from the key space**
- Session IDs
- TLS sessions
- …

Debian: Insecure SSH Key



https://www.debian.org/security/2008/dsa-1571

About Debian Getting Debian Support Developers' Corner

ian / security information / 2008 / security information -- dsa-1571-1 openssl

Debian Security Advisory

DSA-1571-1 openssl -- predictable random number generator

Date Reported:
13 May 2008

Affected Packages:
openssl

Vulnerable:
Yes

Security database references:
In Mitre's CVE dictionary: CVE-2008-0166.

More information:

Luciano Bello discovered that the random number generator in Debian's openssl package is prec
material may be guessable.

What is Randomness?

- Which of the following numbers is random?
  - 01010101
  - 10010110
  - 00000000

What is Randomness?

- Which of the following numbers is random?
  - 01010101
  - 10010110
  - 00000000
- Better question:
  - Have they been produced in a random way?

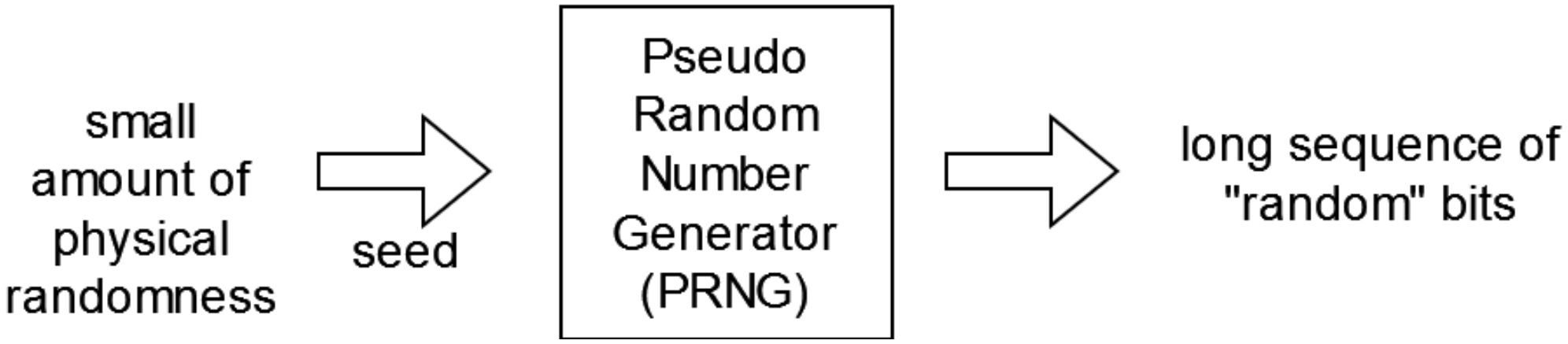
Sources of Randomness

- Physically random events
  - Quantum mechanics, Geiger
  - Thermal noise (built-in into CPU)
  - Key presses and mouse moves?

Random Numbers on a Computer

- There is no algorithm for producing random numbers
- Randomness must be captured from external events
- Difficult to get enough randomness
- => Pseudo-Random Number Generators

Pseudo-Random Number Generator



Two Types of PRNGs

Statistical Pseudo-Random Sequence Generator

- The produced sequence looks random (it passes statistical tests of randomness)
- E.g., Linear Congruential Generators
- Can be used for simulations

Cryptographically Secure Pseudo-Random Sequence Generator

- Unpredictable: Given knowledge of the algorithm and all the previous bits in the keystream

/dev/random Or /dev/urandom?

- `/dev/random`
  - Blocks when insufficient entropy
  - Meant for long-lived keys (e.g., SSH)
- `/dev/urandom`
  - No entropy-requirement applied
  - Meant for short-lived keys (E.g., for TLS connections)
  - Same pool: cryptographic-strength pseudo-random numbers

What about UUIDs?

- Often used as a quick way to generate random strings
- UUIDs version 1 include a high precision timestamp and MAC address -> they are not **random**
- UUIDs version 4 are random. But do not have to be cryptographically secure (according to the RFC)
- **=> Do not blindly use UUIDs if you need cryptographically secure ids**

Example of Insecure Randomness

- Each customer is sent a link to a unique URL containing his receipt:

```
String GenerateReceiptURL(String baseUrl) {
    Random ranGen = new Random();
    ranGen.setSeed((new Date()).getTime());
    return(baseUrl + Gen.nextInt(400000000) + ".html");
}
```

Java: Secure Random Number Generation

- `java.util.Random` unsuitable
  - Based on *linear congruential generator*
    - Sequence is predictable
- Cryptographically secure: `java.security.SecureRandom`
  - Seed is calculated from several entropy sources (e.g., time of I/O events)
  - Not possible to infer from old values to future values

```
import java.security.SecureRandom;
..
Random ranGen = new SecureRandom();
byte[] aesKey = new byte[16]; // 16 bytes = 128 bits
ranGen.nextBytes(aesKey);
```

Developer's Checklist: Random Number Generation in Security Contexts

- **Do not** use a **cryptographically insecure** pseudo random number generator
  - Do not use rand/Random/ packages und function
  - Do not use a Mersenne Twister – based PRNG
  - Do not use a PRNG that is not advertised as being cryptographically secure
- Linux: Use /dev/random or **/dev/urandom** or **getrandom**()
- Windows: Use **BCryptGenRandom**()
- Java: Use **SecureRandom**
- Node.js: **crypto.randomBytes**()

Low- vs. High-level Libraries

Example: Hybrid Encryption

- Goal: Exchange messages securely
  - Encrypt message with random symmetric key
  - Encrypt the symmetric key with asymmetric public key of recipient

Hybrid Encryption with OpenSSL

1. Choose algorithms and parameters, e.g., AES 256 bit, RSA 4096 bit etc.
2. Generate RSA key pair
3. Generate random AES key and nonce
4. Use AES key to encrypt data
5. Hash encrypted data
6. Read RSA private key from wire format
7. Use key to sign hash
8. Read recipient's public key from wire format
9. Use public key to encrypt AES key and signature

Low-level Libraries: Meant for Experts

- It is easy to select wrong parameters
- OpenSSL: Create RSA key pair

```
int RSA_generate_key_ex(RSA *rsa, int bits, BIGNUM *e, BN_GENCB *cb);
```

  - `RSA_generate_key_ex()` generates a key pair and stores it in the RSA structure provided in `rsa`. The pseudo-random number generator must be seeded prior to calling `RSA_generate_key_ex()`.
  - The modulus size will be of length bits, and the public exponent will be e. Key sizes with `num < 1024` should be considered insecure. The exponent is an odd number, typically 3, 17 or 65537.
  - A callback function may be used to provide feedback about the progress of the key generation.

Crypto Fail

SaltStack: RSA e = d = 1

An instance of textbook RSA has three parameters: e, d, and n, where $e = d^{-1} \pmod{\phi(n)}$. Encryption of a message m is m^e, and decryption of a ciphertext c is $c^d = (m^e)^d = m^{ed} = m^1 = m$.

When you set e = 1, encrypting a message does not change it, since raising any number to the power of 1 does not change it. SaltStack was using 1 as their RSA public key (e), so encryption was doing nothing:

```
@@ -47,7 +47,7 @@ def gen_keys(keydir, keyname, keysize, user=None):
    priv = '{0}.pem'.format(base)
    pub = '{0}.pub'.format(base)
-   gen = RSA.gen_key(keysize, 1, callback=lambda x, y, z: None)
+   gen = RSA.gen_key(keysize, 65537, callback=lambda x, y, z: None)
    cumask = os.umask(191)
    gen.save_key(priv, None)
    os.umask(cumask)
```

Good High-level Crypto Libraries

- libsodium
  - C, included in standard PHP library
  - Bindings for other programming languages
  - Opinionated library
    - Algorithms have been selected and cannot be changed
  - https://libsodium.org
- Tink
  - https://github.com/google/tink
  - For: Java, C++, Obj-C, Go, Python

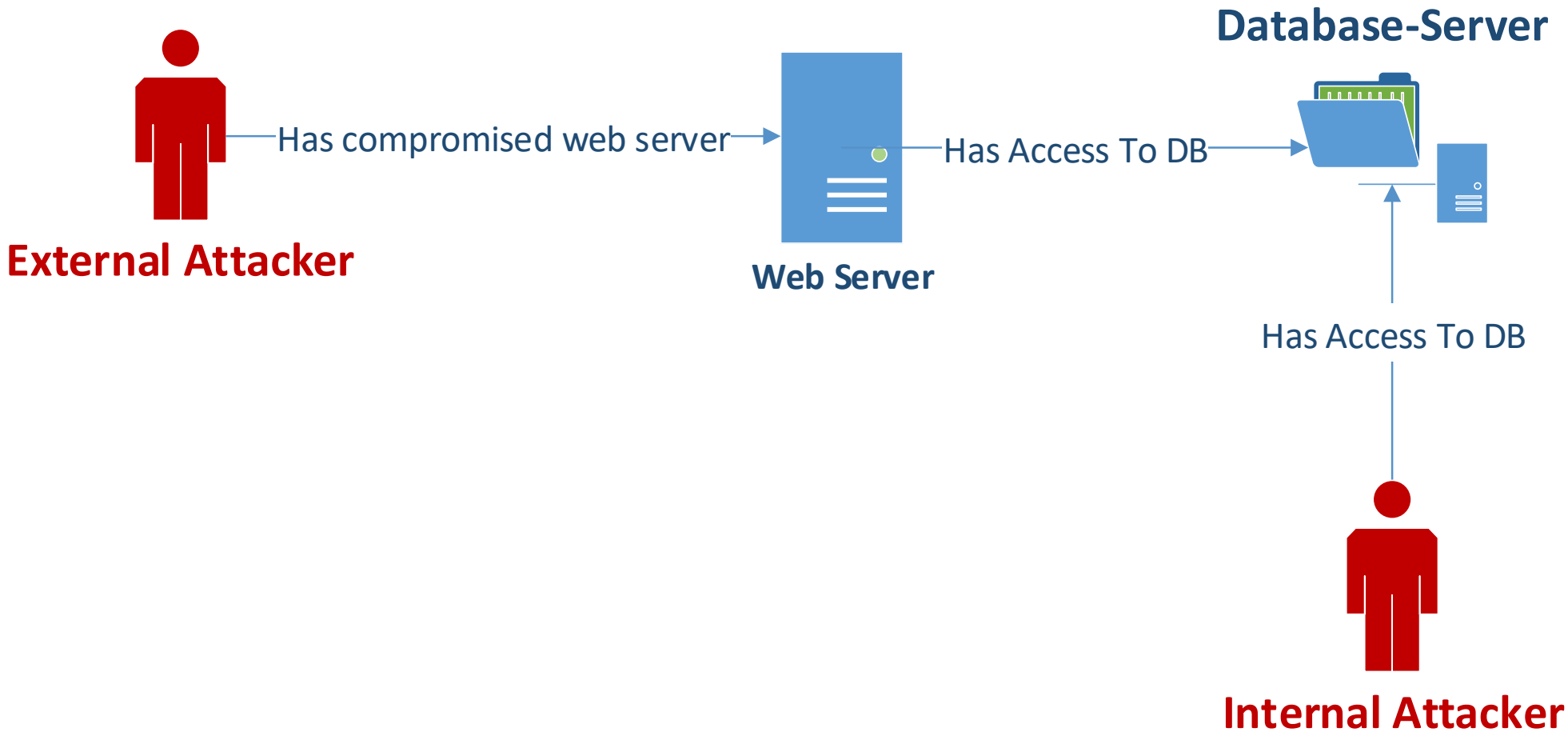
libsodium: Generate Keypair (PHP)

```php
$keypair = sodium_crypto_box_keypair();
$publicKey = sodium_crypto_box_publickey($keypair);
$secretKey = sodium_crypto_box_secretkey($keypair);
```

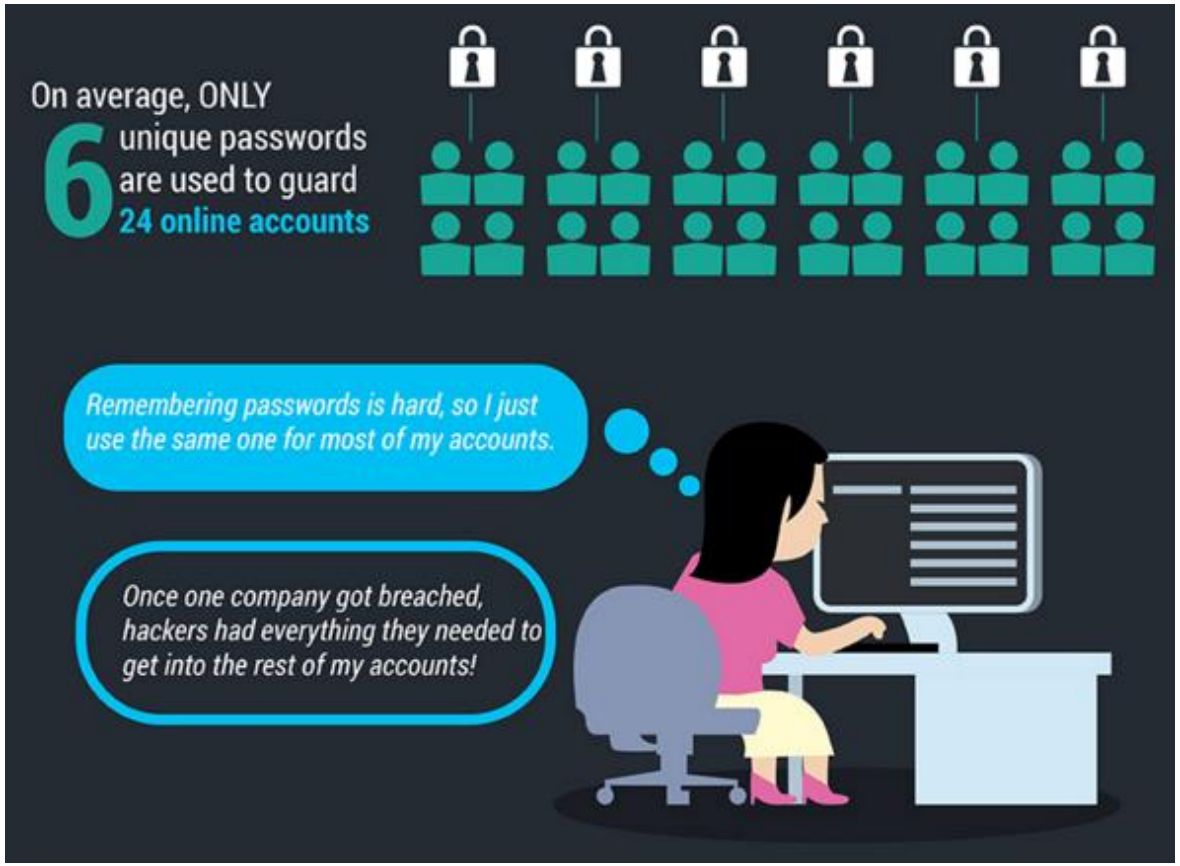
- The `sodium_crypto_box_keypair()` function randomly generates a secret key and a corresponding public key with modern algorithms. (currently X25519)

Storing Passwords

Attack Scenario



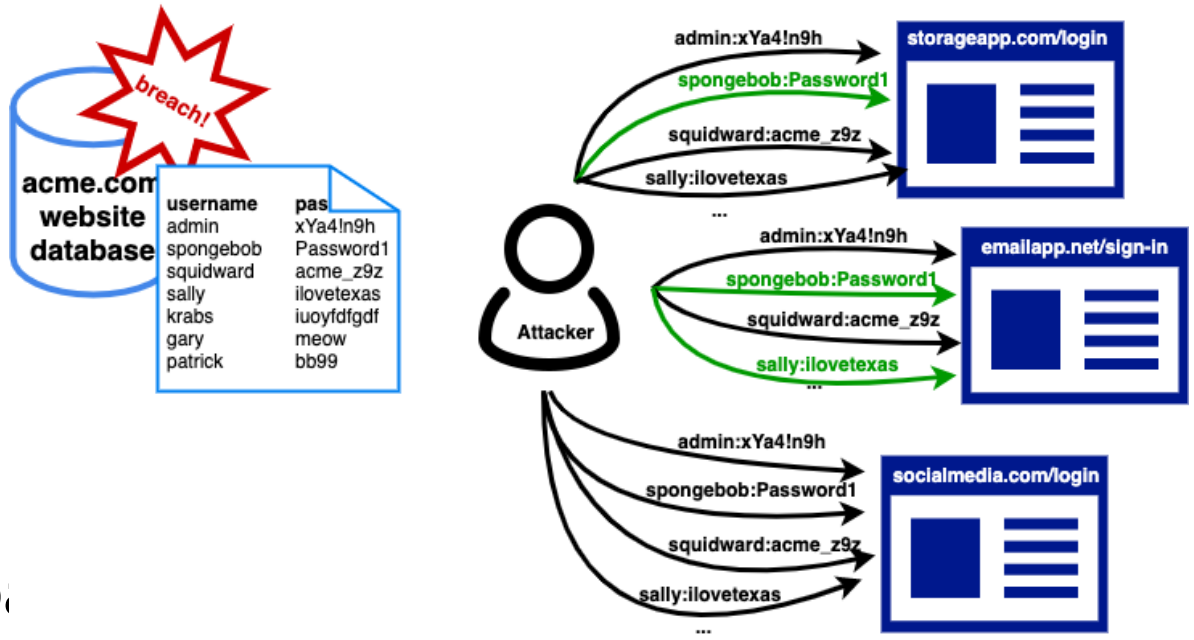
Threat: Password Reuse



Source: https://www.vericlouds.com/stolen-credentials-hackers-breach-secure-organizations/

Credential Stuffing

- Brute-force logins based on list of leaked credentials
- Same username/email and password is tested across websites

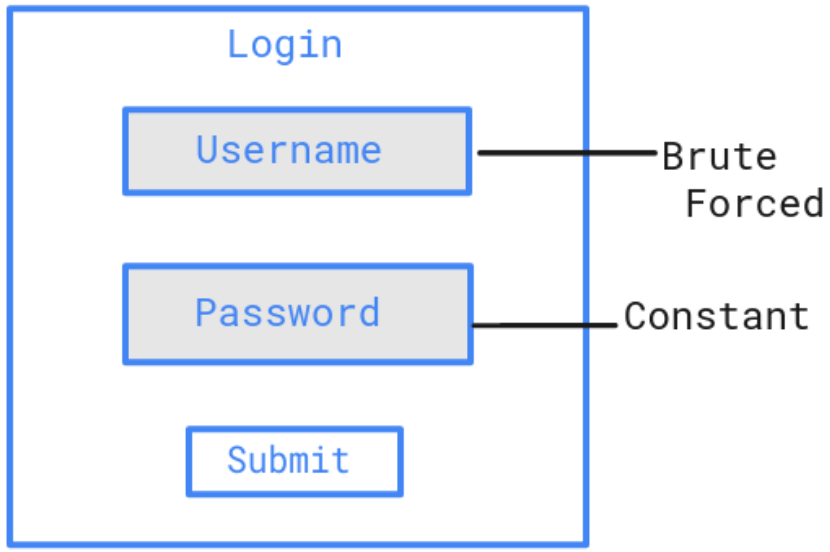


- Exploits: p

Source: https://owasp.org/www-community/attacks/Credential_stuffing

Password Spraying

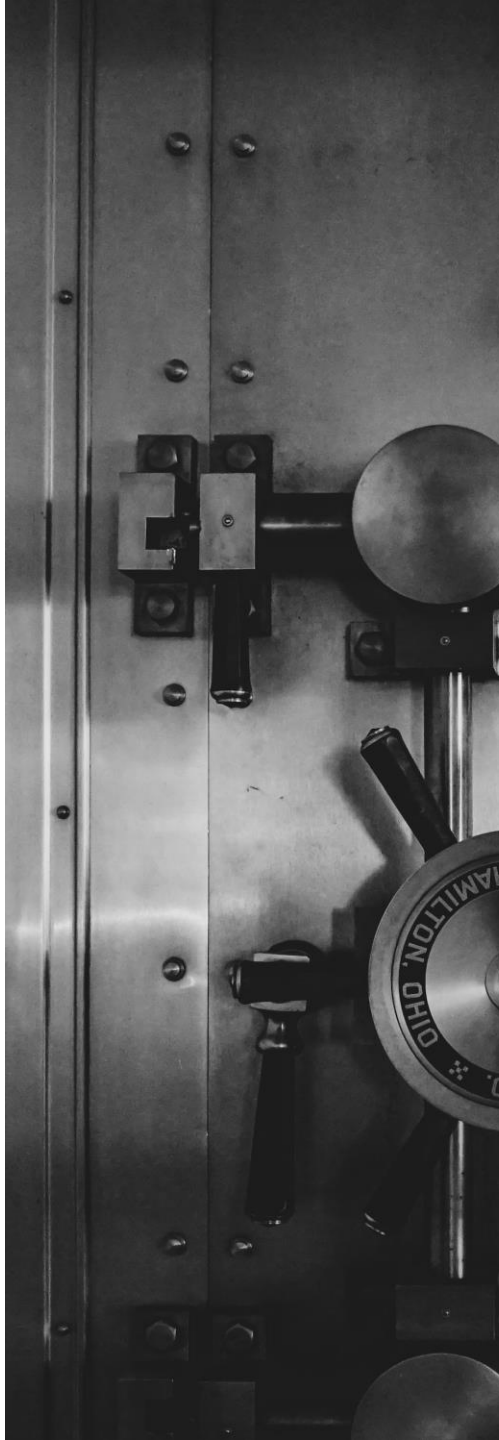
- Brute-force logins based on list of usernames and common passwords
- Attacks ma default)
  password t
- Exploits:
  Weak/Defa
  passwords



Source: https://owasp.org/www-community/attacks/Password_Spraying_Attack

A Short History

- Since the 70s passwords stored as hash with an individual salt
- Computers became faster
  - Brute-Forcing became easier
  - Defense => More iterations
- In the last 10 years:
  - Attackers use FPGAs, dedicated ASIC modules
  - Multi core GPU
  - Defense => Memory Hard functions



Hashcat

- Most popular tool used by password crackers
  - GPU-support
  - before: John the ripper
  - **fast**
- Multi-OS (Linux, Windows and macOS)
- Open-source
  - https://hashcat.net/hashcat/
- Based on the OpenCL Runtime
  - GPU, CPU
- 200+ Hash-types implemented

Which Hashing Algorithm?

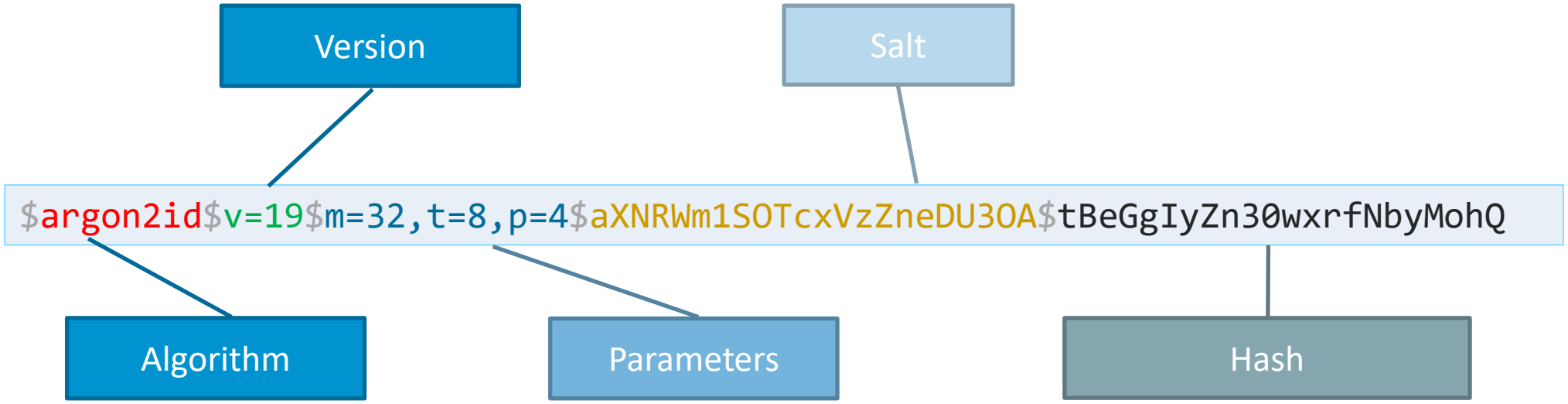
OWASP Password_Storage_Cheat_Sheet:

1. **Argon2id**
   - First choice
2. Scrypt
   - If Argon2id is not available
3. bcrypt
   - For legacy systems when no other algorithm support available
4. PBKDF2
   - For FIPS-140 compliance

Argon2

- Winner of the Password Hashing Competition (PHC) in July 2015
  - Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich from the University of Luxembourg
- RFC: https://www.rfc-editor.org/rfc/rfc9106.html
- State of the art memory-hard function
- Argon2 is optimized for the x86 architecture
- Reference C Implementation: https://github.com/P-H-C/phc-winner-argon2/
  - Bindings exist for many languages

Argon2id in modular crypt format



Argon2: How to Select Parameters

- Tradeoff between security and usability
  - Higher values => more security
  - Higher values => more time need, less responsive
- According to OWASP
  - Memory **m**: 15MiB memory
  - Parallelism **p: 1**
  - Number of iterations **t: 2**
- As a rule: Calculating a hash should take roughly 1 second

Password Hashing with libsodium (PHP)

- On user creation: store an ASCII encoded string in

```
sodium_crypto_pwhash_str(string $password,
                         int $opslimit,
                         int $memlimit): string
```

  - memory-hard, CPU-intensive hash function
    - currently Argon2id
  - Automatically generated salt
  - Algorithm and salt are part of the result

Password Hashing with libsodium (PHP)

- On user login: Verifying the password

```
sodium_crypto_pwhash_str_verify(string $hash,
                                string $password): bool
```

  - `$hash` would be the hash from the DB
    - Contains salt, t and memory requirements
  - `$password` the user input

Cryptography: Important Rules

- ❑ **Don’t roll your own crypto!**
- ❑ Easy-to-use libraries (libsodium)
- ❑ Don’t just check the “encryption” checkbox – be fully aware of the threats and whether crypto can help!
- ❑ Include good key management in the design
- ❑ Use good randomness
- ❑ Use AEAD ciphers for integrity in symmetric crypto
- ❑ Use expensive key derivation when the key base is a human-generated password (Argon2id or PBKDF2)
- ❑ **Good applied crypto is hard – get help if necessary!**

Quiz!

Kahoot!

Applied Cryptography

Visit Kahoot and enter the game pin I'll share with you!

https://kahoot.it

Learning Resources

Where to learn more about secure software development

Understanding is Key

You won‘t get software security for free

- Cultivate a culture of continuous learning
- Understand your language, runtime, platform, IDE, build tools, relevant vulnerability classes, threats
- Develop by-design countermeasures
- Simplify and reduce

Where Can I Learn Software Security?

General

- Testing and securing your own software!
- sec4dev
- Security Meetup by SBA Research

https://www.meetup.com/de-DE/Security-Meetup-by-SBA-Research/

Where Can I Learn Software Security?

Web Security

- PortSwigger Web Academy
- OWASP Juice Shop
- OWASP Cheat Sheets: https://cheatsheetseries.owasp.org/index.html
- OWASP Application Security Verification Standard (ASVS)

Thank You!

Ulrich Bayer

SBA Research

Floragasse 7, 1040 Vienna, Austria

E-mail: ubayer@sba-research.org

Matrix: @ubayer:sba-research.org