Edinburgh Napier
UNIVERSITY

Secure Software Development

Introduction

Prof Ashkan Sami

Arian 5 rocket launch - Why Did It Fail?

- **Software Error in Inertial Reference System (IRS)**
  - Code was reused from Ariane 4 without checking new flight dynamics.
  - Ariane 4’s code assumed a lower velocity range.
  - Ariane 5’s faster speeds broke those assumptions.
  - Conversation of 64 bit into 16-bit integer caused an integer overflow!
- **Unhandled Exception**
  - Overflow generated an exception.
  - The system didn’t degrade gracefully.
  - System shut down, and backup system (same software) also failed.

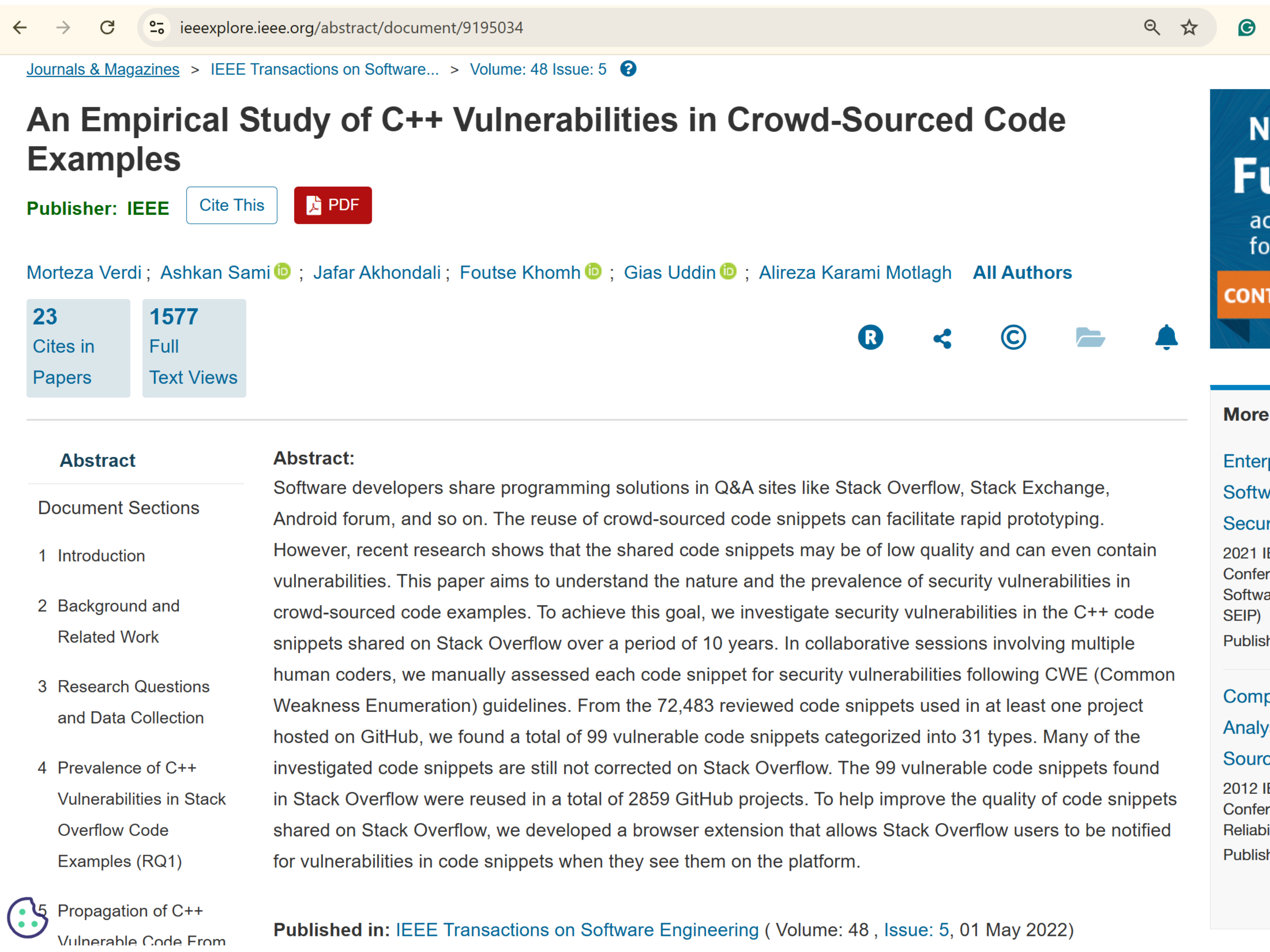
00	00	12	34
		12	34

00	2H	12	34
		12	34

Professor Ashkan Sami

- BS from Virginia Tech; U.S.
- MSc in Artificial Intelligence and Robotics; Shiraz University
- PhD from Tohoku University; Japan
- Assistant Professor at Tohoku University; Japan until 2008
- Professor at Edinburgh Napier University **a.sami@napier.ac.uk**
- Lead of SICSA on Systems and Software Engineering Research theme: https://www.sicsa.ac.uk/research/networking-and-systems/
  - SICSA member institutions comprising all 14 Scottish Higher Education Computer Science & Informatics Schools and Departments and three Scottish Innovation Centres

- **Some Highlights of my work:**



An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples

Publisher: IEEE

Morteza Verdi ; Ashkan Sami ; Jafar Akhondali ; Foutse Khomh ; Gias Uddin ; Alireza Karami Motlagh **All Authors**

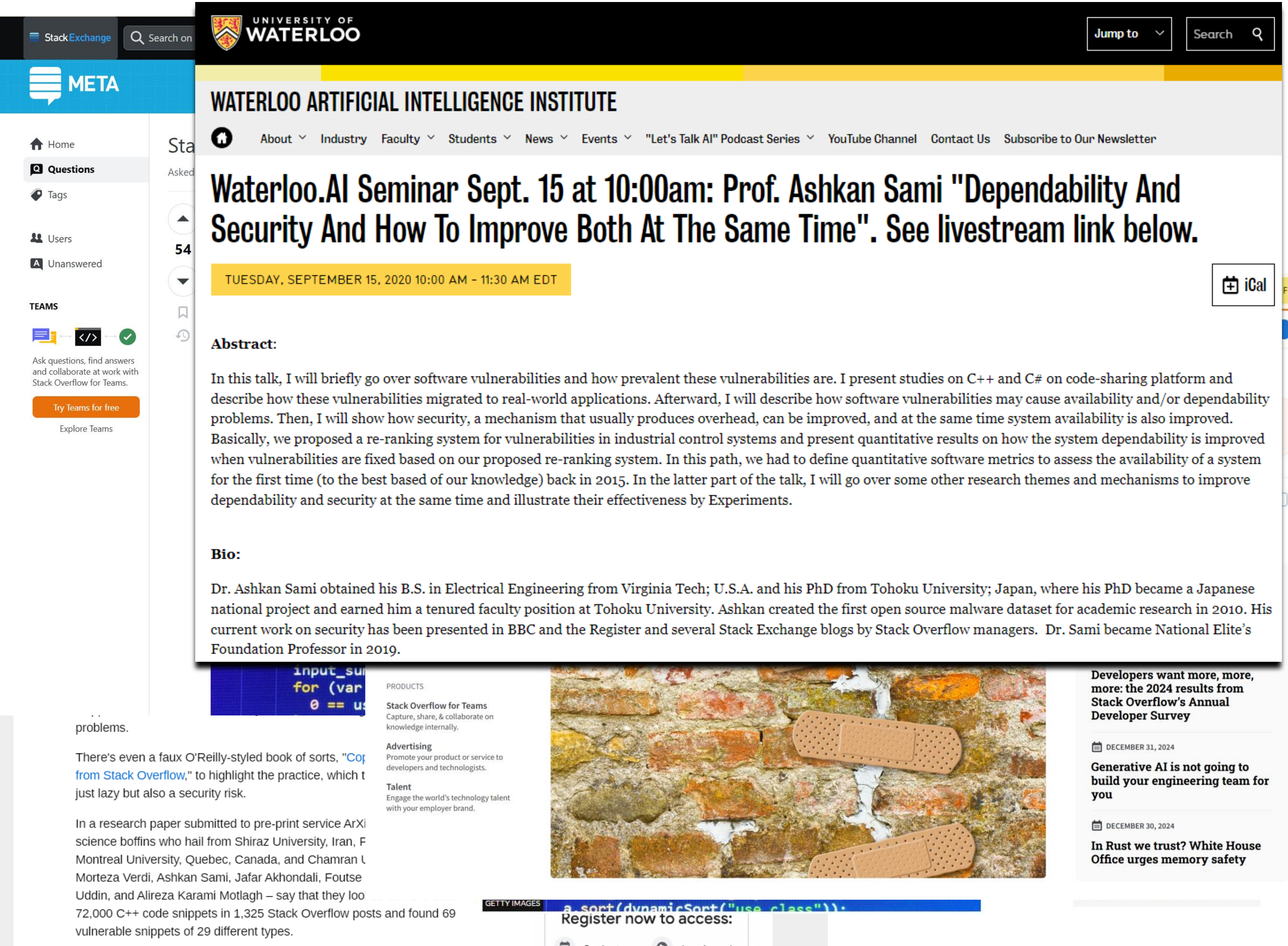
23 Cites in Papers

1577 Full Text Views

Abstract:

Software developers share programming solutions in Q&A sites like Stack Overflow, Stack Exchange, Android forum, and so on. The reuse of crowd-sourced code snippets can facilitate rapid prototyping. However, recent research shows that the shared code snippets may be of low quality and can even contain vulnerabilities. This paper aims to understand the nature and the prevalence of security vulnerabilities in crowd-sourced code examples. To achieve this goal, we investigate security vulnerabilities in the C++ code snippets shared on Stack Overflow over a period of 10 years. In collaborative sessions involving multiple human coders, we manually assessed each code snippet for security vulnerabilities following CWE (Common Weakness Enumeration) guidelines. From the 72,483 reviewed code snippets used in at least one project hosted on GitHub, we found a total of 99 vulnerable code snippets categorized into 31 types. Many of the investigated code snippets are still not corrected on Stack Overflow. The 99 vulnerable code snippets found in Stack Overflow were reused in a total of 2859 GitHub projects. To help improve the quality of code snippets shared on Stack Overflow, we developed a browser extension that allows Stack Overflow users to be notified for vulnerabilities in code snippets when they see them on the platform.

Published in: IEEE Transactions on Software Engineering (Volume: 48 , Issue: 5, 01 May 2022)



WATERLOO ARTIFICIAL INTELLIGENCE INSTITUTE

Waterloo.AI Seminar Sept. 15 at 10:00am: Prof. Ashkan Sami "Dependability And Security And How To Improve Both At The Same Time". See livestream link below.

TUESDAY, SEPTEMBER 15, 2020 10:00 AM - 11:30 AM EDT

Abstract:

In this talk, I will briefly go over software vulnerabilities and how prevalent these vulnerabilities are. I present studies on C++ and C# on code-sharing platform and describe how these vulnerabilities migrated to real-world applications. Afterward, I will describe how software vulnerabilities may cause availability and/or dependability problems. Then, I will show how security, a mechanism that usually produces overhead, can be improved, and at the same time system availability is also improved. Basically, we proposed a re-ranking system for vulnerabilities in industrial control systems and present quantitative results on how the system dependability is improved when vulnerabilities are fixed based on our proposed re-ranking system. In this path, we had to define quantitative software metrics to assess the availability of a system for the first time (to the best based of our knowledge) back in 2015. In the latter part of the talk, I will go over some other research themes and mechanisms to improve dependability and security at the same time and illustrate their effectiveness by Experiments.

Bio:

Dr. Ashkan Sami obtained his B.S. in Electrical Engineering from Virginia Tech; U.S.A. and his PhD from Tohoku University; Japan, where his PhD became a Japanese national project and earned him a tenured faculty position at Tohoku University. Ashkan created the first open source malware dataset for academic research in 2010. His current work on security has been presented in BBC and the Register and several Stack Exchange blogs by Stack Overflow managers. Dr. Sami became National Elite's Foundation Professor in 2019.

Ask questions, find answers and collaborate at work with Stack Overflow for Teams.

```
input_su
for (var
0 == u
```

PRODUCTS

Stack Overflow for Teams
Capture, share, & collaborate on knowledge internally.

Advertising
Promote your product or service to developers and technologists.

Talent
Engage the world's technology talent with your employer brand.

Developers want more, more, more: the 2024 results from Stack Overflow's Annual Developer Survey

DECEMBER 31, 2024
Generative AI is not going to build your engineering team for you

DECEMBER 30, 2024
In Rust we trust? White House Office urges memory safety

problems.

There's even a faux O'Reilly-styled book of sorts, "Co from Stack Overflow," to highlight the practice, which t just lazy but also a security risk.

In a research paper submitted to pre-print service ArXi science boffins who hail from Shiraz University, Iran, F Montreal University, Quebec, Canada, and Chamran U Morteza Verdi, Ashkan Sami, Jafar Akhondali, Foutse Uddin, and Alireza Karami Motlagh – say that they loo 72,000 C++ code snippets in 1,325 Stack Overflow posts and found 69 vulnerable snippets of 29 different types.

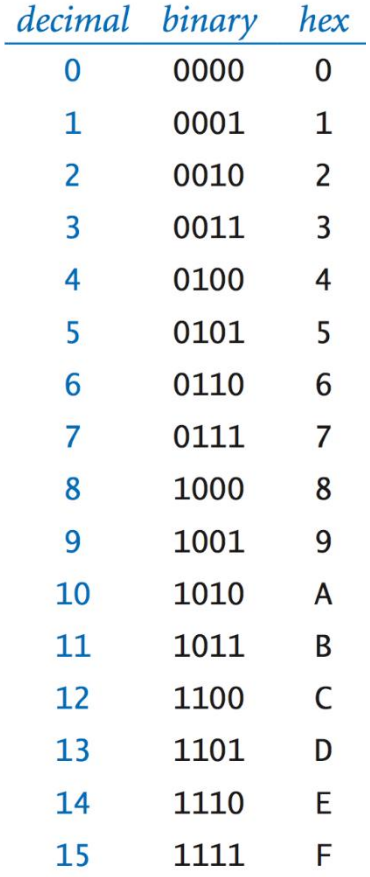
```
a.sort(dynamicSort("use class"));
```

Register now to access:

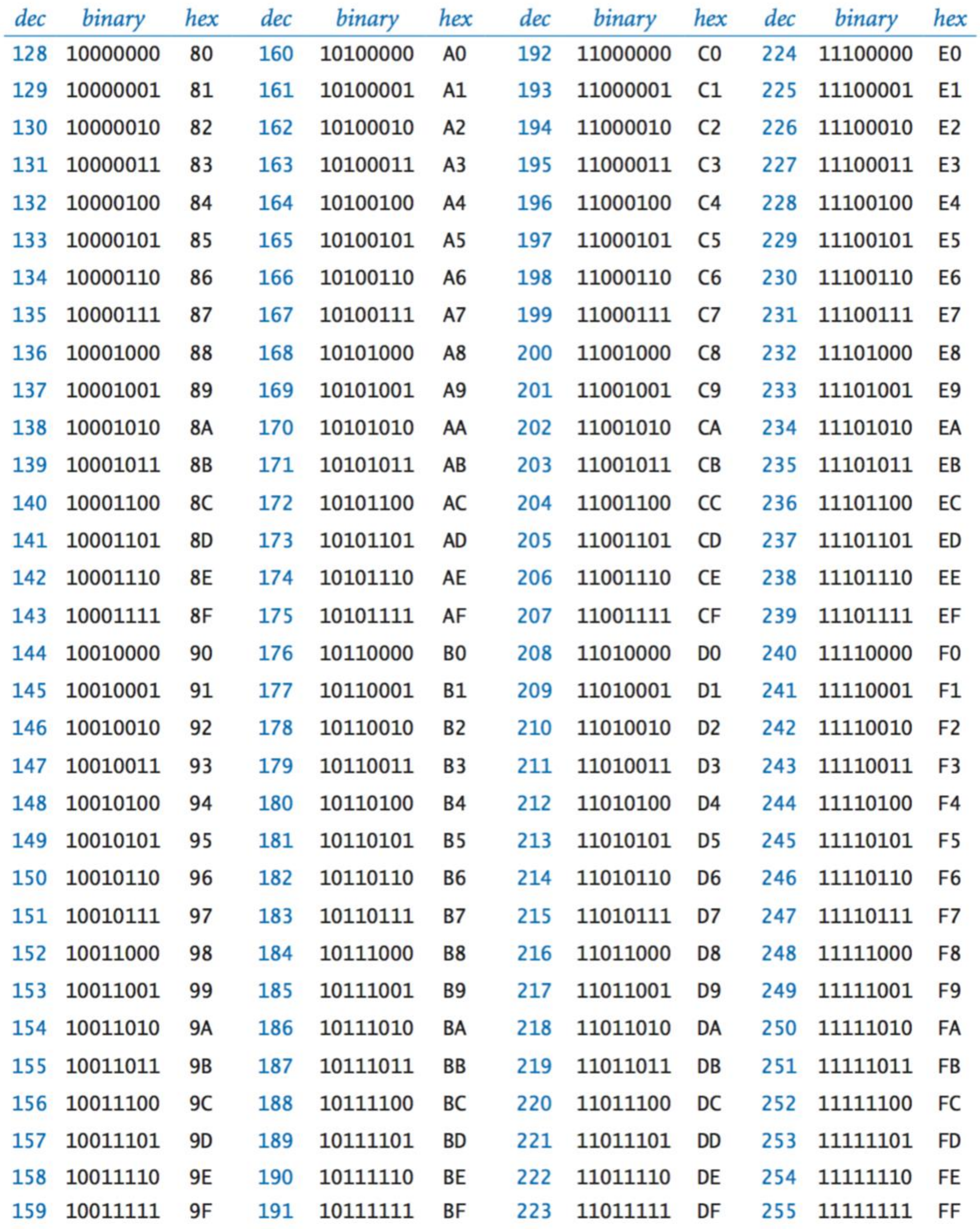
Contents

- Some definitions
- Building secure systems
- Policy, Threat Models, and Mechanisms
- CIA (Confidentiality, Integrity and Availability)
- CWEs
- Hard Coded-Credentials
- Overview of some other CWEs

Binary representation of number

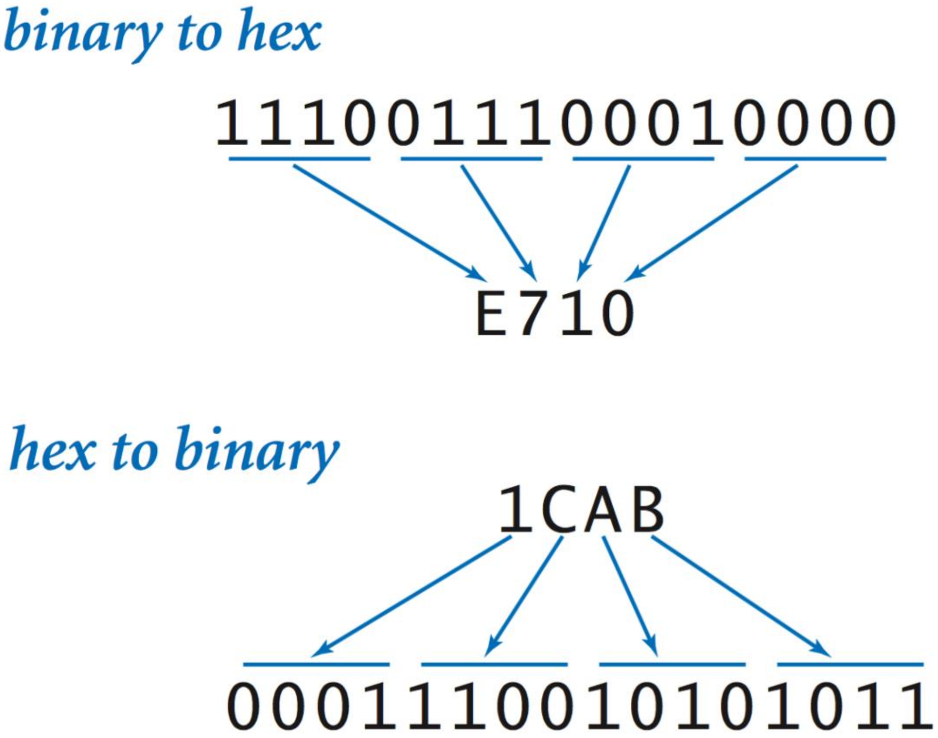


decimal	binary	hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



dec	binary	hex	dec	binary	hex	dec	binary	hex	dec	binary	hex
128	10000000	80	160	10100000	A0	192	11000000	C0	224	11100000	E0
129	10000001	81	161	10100001	A1	193	11000001	C1	225	11100001	E1
130	10000010	82	162	10100010	A2	194	11000010	C2	226	11100010	E2
131	10000011	83	163	10100011	A3	195	11000011	C3	227	11100011	E3
132	10000100	84	164	10100100	A4	196	11000100	C4	228	11100100	E4
133	10000101	85	165	10100101	A5	197	11000101	C5	229	11100101	E5
134	10000110	86	166	10100110	A6	198	11000110	C6	230	11100110	E6
135	10000111	87	167	10100111	A7	199	11000111	C7	231	11100111	E7
136	10001000	88	168	10101000	A8	200	11001000	C8	232	11101000	E8
137	10001001	89	169	10101001	A9	201	11001001	C9	233	11101001	E9
138	10001010	8A	170	10101010	AA	202	11001010	CA	234	11101010	EA
139	10001011	8B	171	10101011	AB	203	11001011	CB	235	11101011	EB
140	10001100	8C	172	10101100	AC	204	11001100	CC	236	11101100	EC
141	10001101	8D	173	10101101	AD	205	11001101	CD	237	11101101	ED
142	10001110	8E	174	10101110	AE	206	11001110	CE	238	11101110	EE
143	10001111	8F	175	10101111	AF	207	11001111	CF	239	11101111	EF
144	10010000	90	176	10110000	B0	208	11010000	D0	240	11110000	F0
145	10010001	91	177	10110001	B1	209	11010001	D1	241	11110001	F1
146	10010010	92	178	10110010	B2	210	11010010	D2	242	11110010	F2
147	10010011	93	179	10110011	B3	211	11010011	D3	243	11110011	F3
148	10010100	94	180	10110100	B4	212	11010100	D4	244	11110100	F4
149	10010101	95	181	10110101	B5	213	11010101	D5	245	11110101	F5
150	10010110	96	182	10110110	B6	214	11010110	D6	246	11110110	F6
151	10010111	97	183	10110111	B7	215	11010111	D7	247	11110111	F7
152	10011000	98	184	10111000	B8	216	11011000	D8	248	11111000	F8
153	10011001	99	185	10111001	B9	217	11011001	D9	249	11111001	F9
154	10011010	9A	186	10111010	BA	218	11011010	DA	250	11111010	FA
155	10011011	9B	187	10111011	BB	219	11011011	DB	251	11111011	FB
156	10011100	9C	188	10111100	BC	220	11011100	DC	252	11111100	FC
157	10011101	9D	189	10111101	BD	221	11011101	DD	253	11111101	FD
158	10011110	9E	190	10111110	BE	222	11011110	DE	254	11111110	FE
159	10011111	9F	191	10111111	BF	223	11011111	DF	255	11111111	FF

Internal Representation of Numbers



binary to hex

1110011100010000

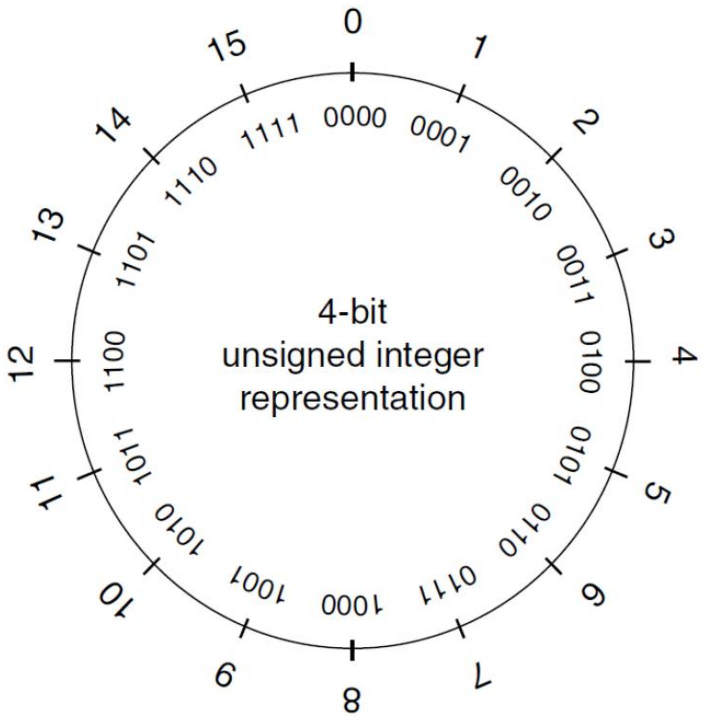
E710

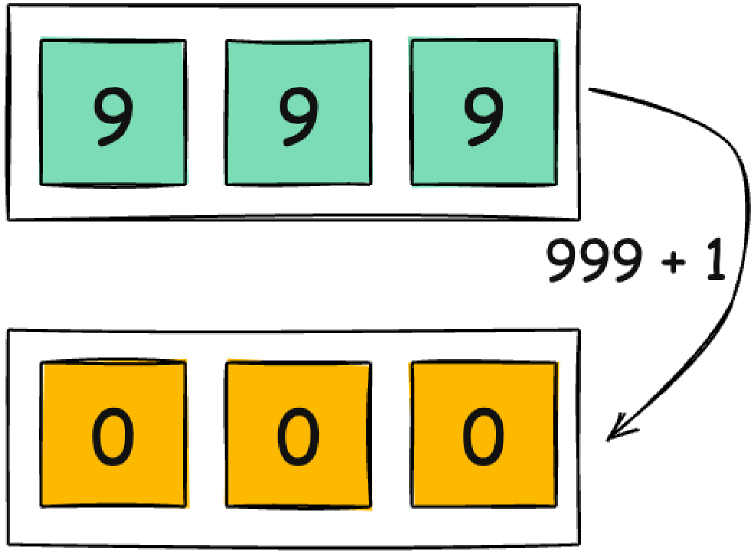
hex to binary

1CAB

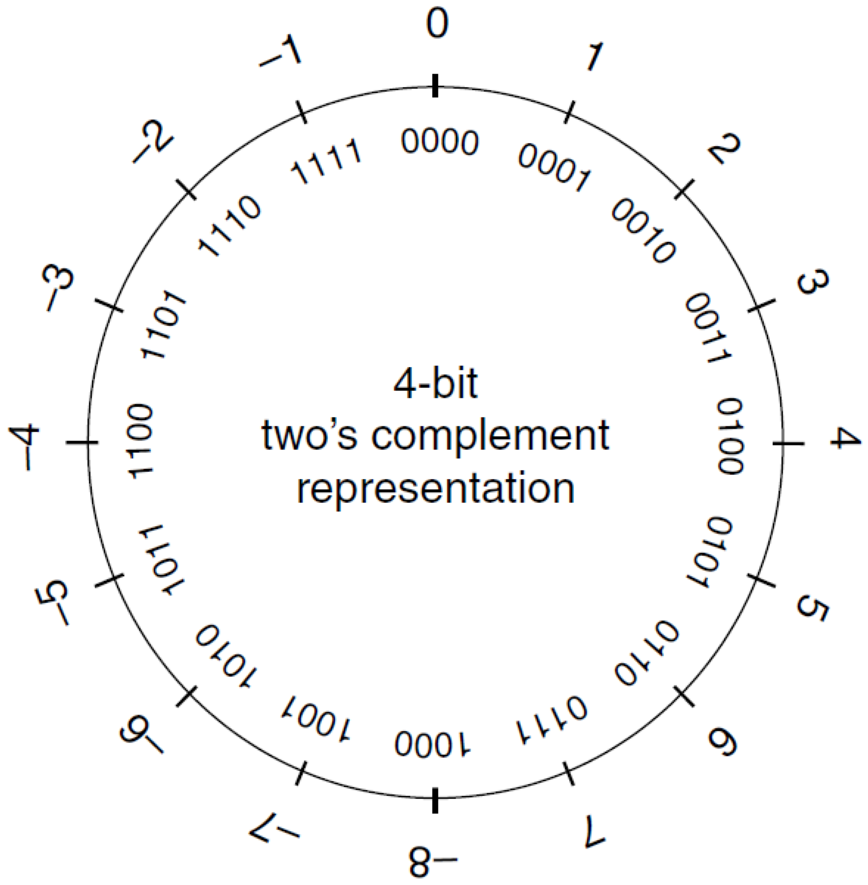
0001110010101011

Four-bit unsigned Integer presentation

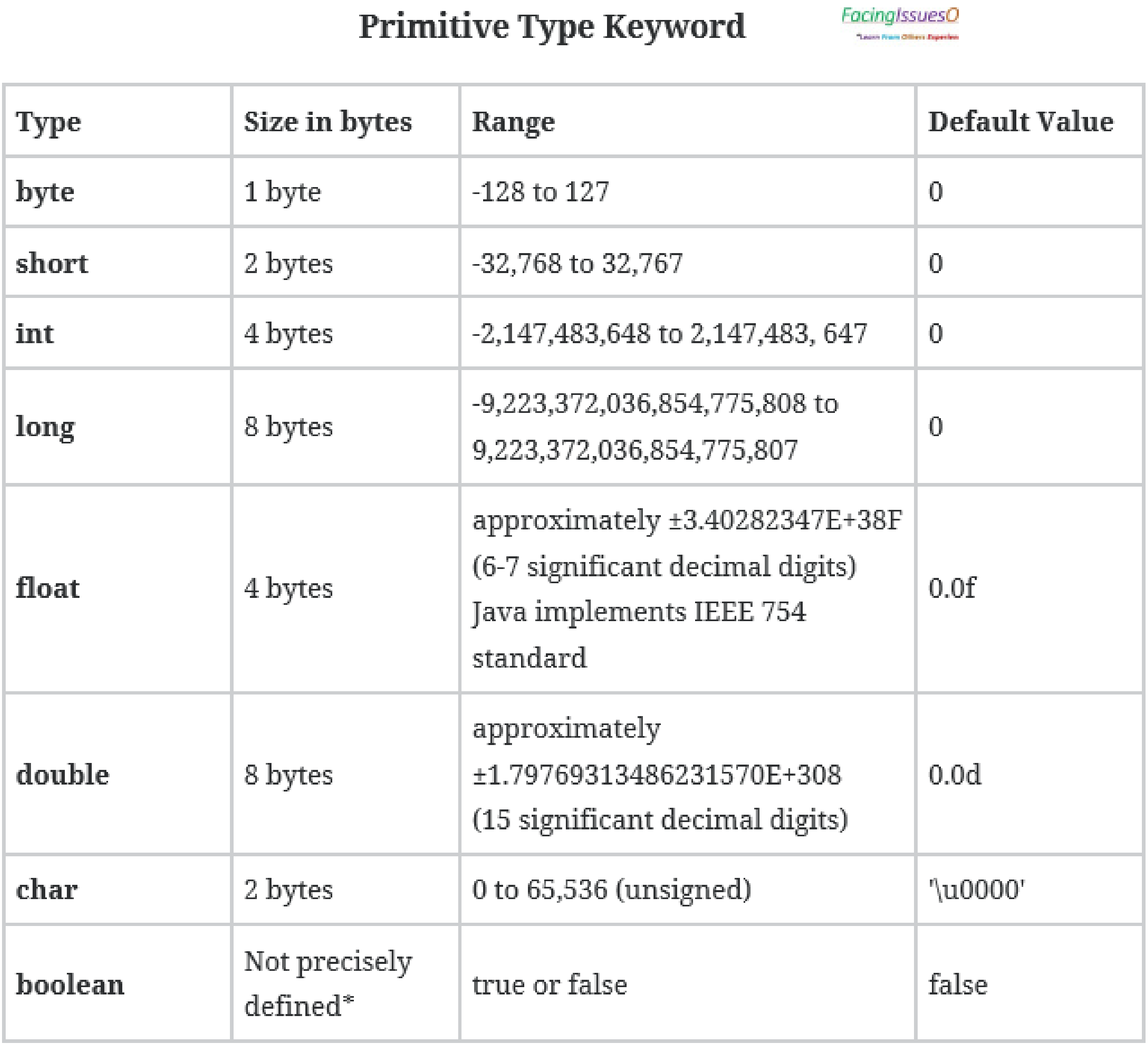




Four-bit signed Integer presentation



Binary	Decimal	Weighting
00000000	0	0
00000001	1	2^0
01111110	126	$2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1$
01111111	127	$2^{N-1} - 1$
10000000	–128	$-(2^{N-1}) + 0$
10000001	–127	$-(2^{N-1}) + 1$
11111110	–2	$-(2^{N-1}) + 126$
11111111	–1	$-(2^{N-1}) + 127$

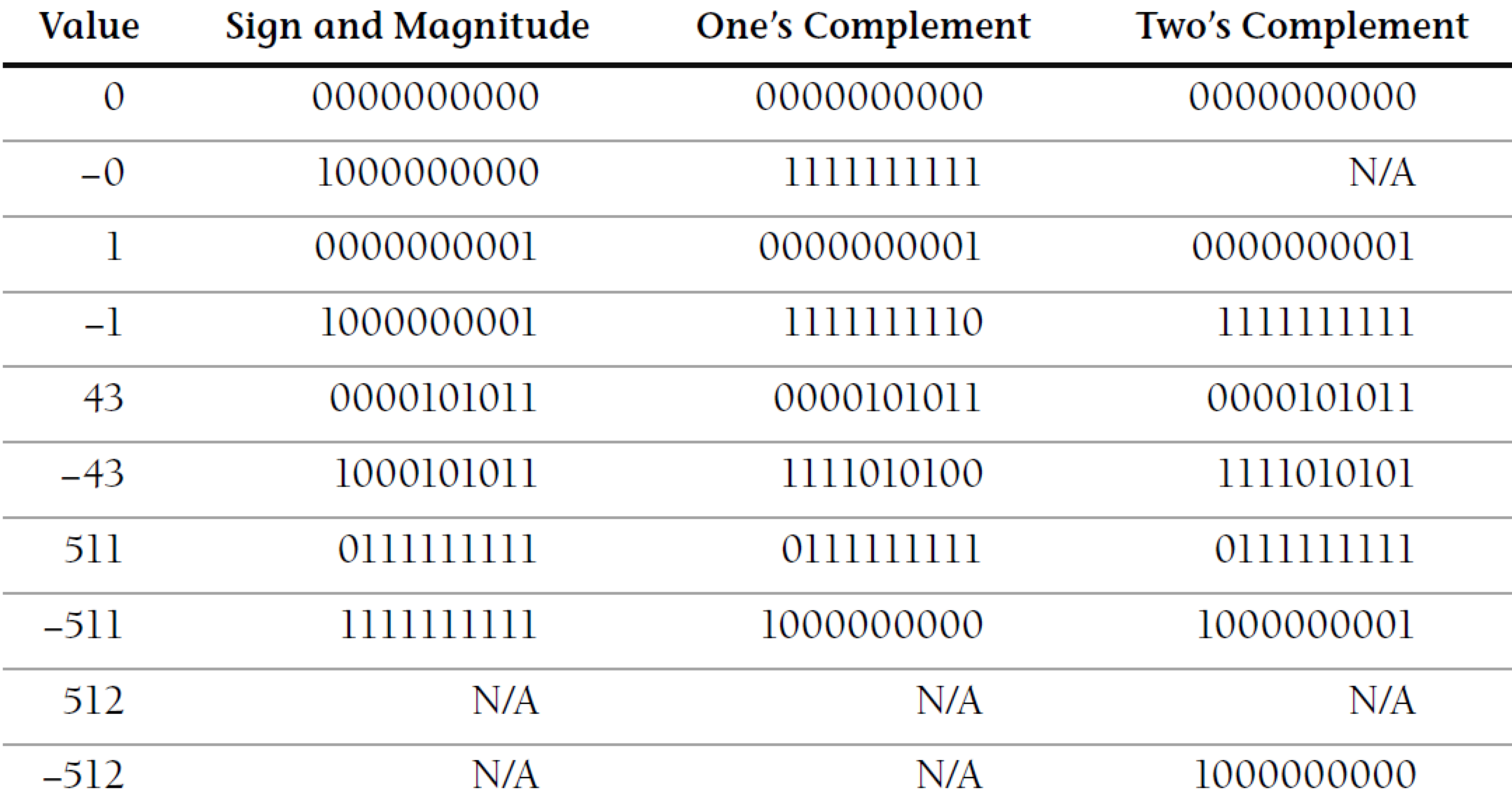


Primitive Type Keyword

FacingIssuesOnIT

Type	Size in bytes	Range	Default Value
byte	1 byte	-128 to 127	0
short	2 bytes	-32,768 to 32,767	0
int	4 bytes	-2,147,483,648 to 2,147,483, 647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard	0.0f
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)	0.0d
char	2 bytes	0 to 65,536 (unsigned)	'\u0000'
boolean	Not precisely defined*	true or false	false

https://facingissuesonit.com/2019/06/24/java-primitive-type/



Value	Sign and Magnitude	One's Complement	Two's Complement
0	0000000000	0000000000	0000000000
–0	1000000000	1111111111	N/A
1	0000000001	0000000001	0000000001
–1	1000000001	1111111110	1111111111
43	0000101011	0000101011	0000101011
–43	1000101011	1111010100	1111010101
511	0111111111	0111111111	0111111111
–511	1111111111	1000000000	1000000001
512	N/A	N/A	N/A
–512	N/A	N/A	1000000000

Definitions

- summary of the 8 primitive data types in Java:

1. **byte**:
   1. Description: 8-bit signed two's complement integer.
   2. Range: -128 to 127 (inclusive).
   3. Storage: 1 byte.
2. **short**:
   1. Description: 16-bit signed two's complement integer.
   2. Range: -32,768 to 32,767 (inclusive).
   3. Storage: 2 bytes.
3. **int**:
   1. Description: 32-bit signed two's complement integer.
   2. Range: -2^31 to 2^31-1 (about -2.1 billion to 2.1 billion).
   3. Storage: 4 bytes.

Definitions

- summary of the eight primitive data types in Java:

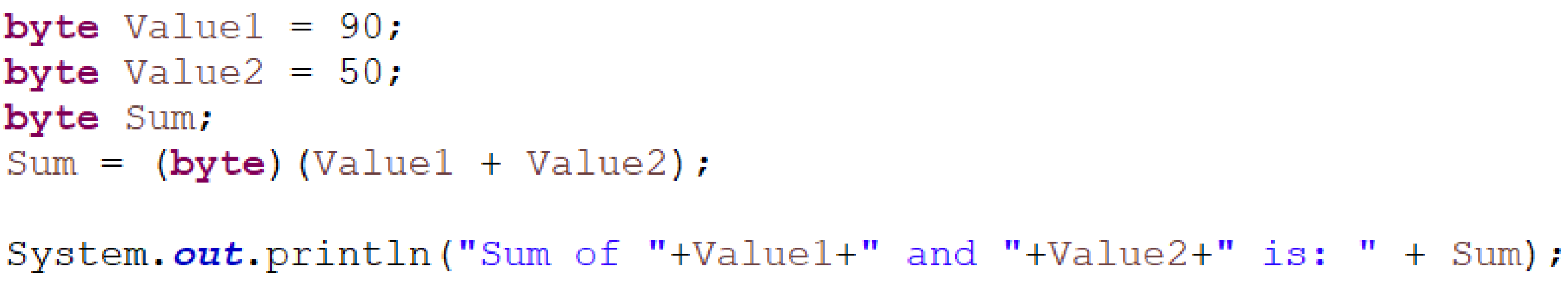
4. **long**:
   1. Description: 64-bit signed two's complement integer.
   2. Range: -2^63 to 2^63-1 (about -9.2 quintillion to 9.2 quintillion).
   3. Storage: 8 bytes.
5. **float**:
   1. Description: Single-precision 32-bit IEEE 754 floating point.
   2. Range: Approximately ±3.40282347E+38F (6-7 significant decimal digits).
   3. Storage: 4 bytes.
6. **double**:
   1. Description: Double-precision 64-bit IEEE 754 floating point.
   2. Range: Approximately ±1.79769313486231570E+308 (15 significant decimal digits).
   3. Storage: 8 bytes.

Definitions

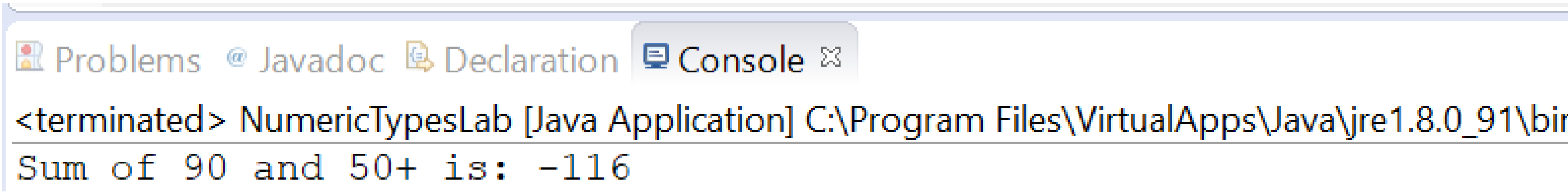
- summary of the eight primitive data types in Java:

7. **char**:
    1. Description: A single 16-bit Unicode character.
    2. Range: 0 to 65,535 (inclusive).
    3. Storage: 2 bytes.
8. **boolean**:
    1. Description: Represents one bit of information, but its "size" isn't precisely defined.
    2. Range: true or false.
    3. Storage: Not precisely defined, typically depends on the Java Virtual Machine (JVM) implementation. Often 1 byte is used for practicality, but it can vary.

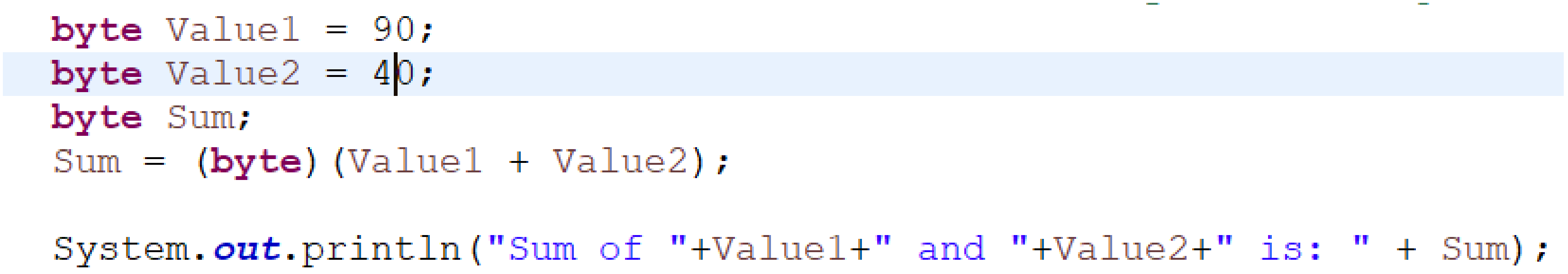
Now let’s try some of these in action!



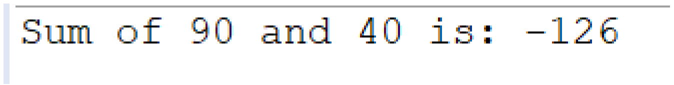
```
byte Value1 = 90;
byte Value2 = 50;
byte Sum;
Sum = (byte)(Value1 + Value2);

System.out.println("Sum of "+Value1+" and "+Value2+" is: " + Sum);
```



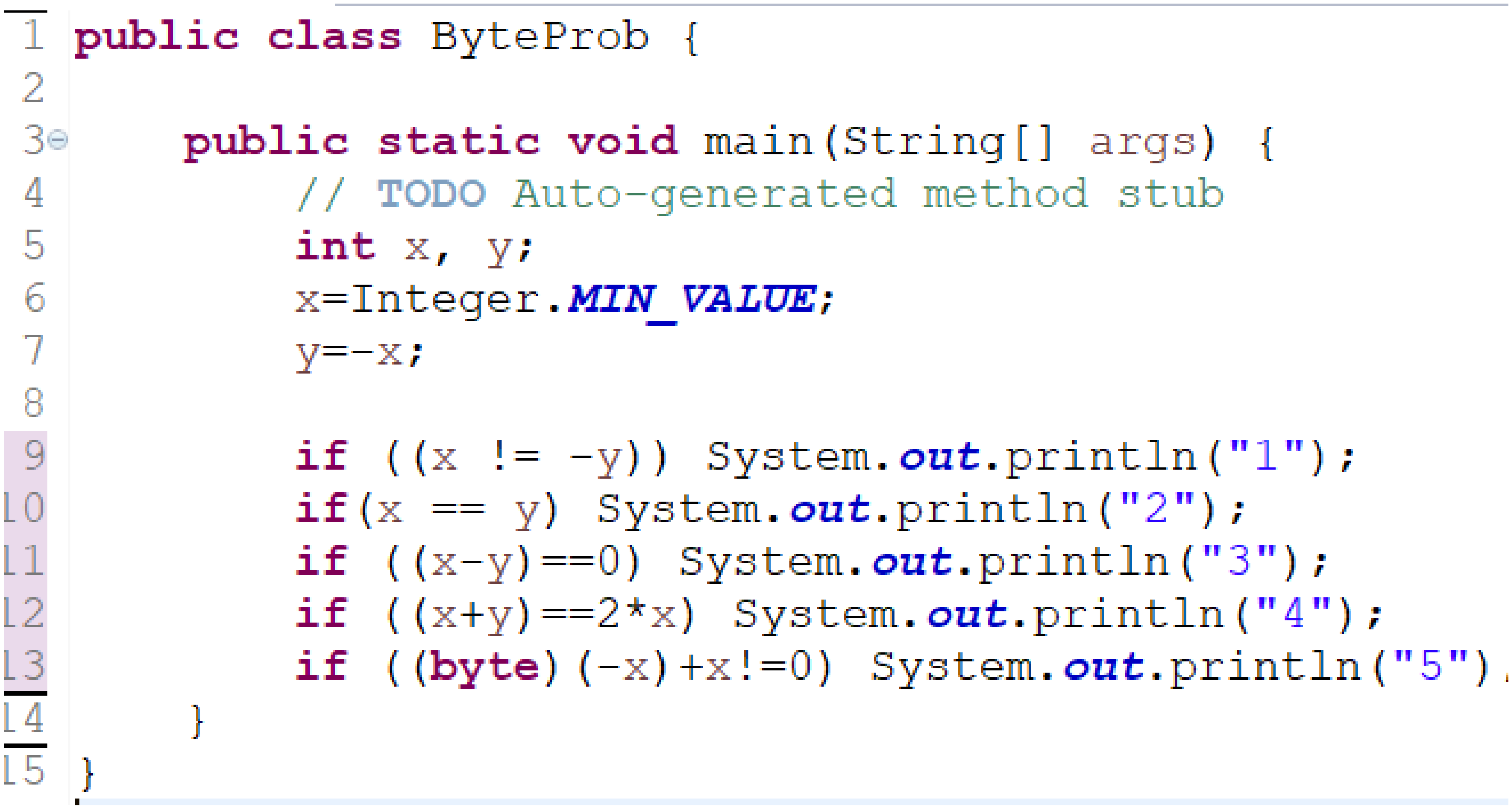
```
<terminated> NumericTypesLab [Java Application] C:\Program Files\VirtualApps\Java\jre1.8.0_91\bi
Sum of 90 and 50+ is: -116
```



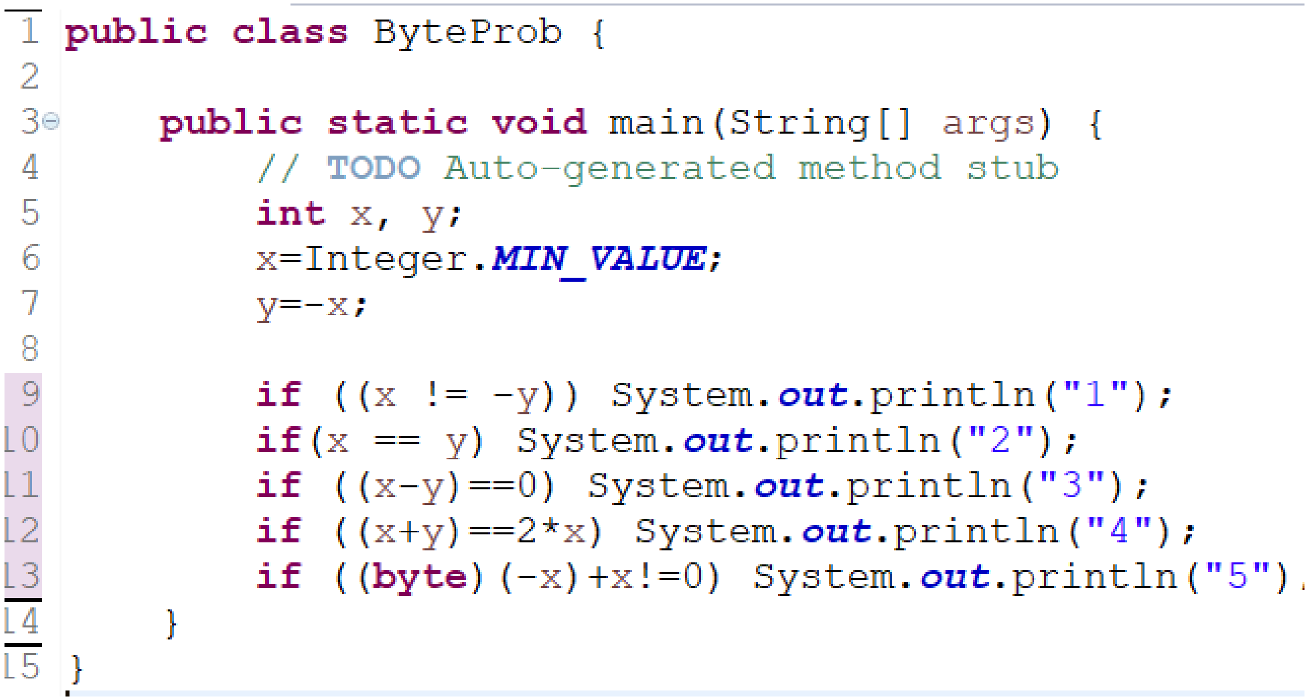
```
byte Value1 = 90;
byte Value2 = 40;
byte Sum;
Sum = (byte)(Value1 + Value2);

System.out.println("Sum of "+Value1+" and "+Value2+" is: " + Sum);
```



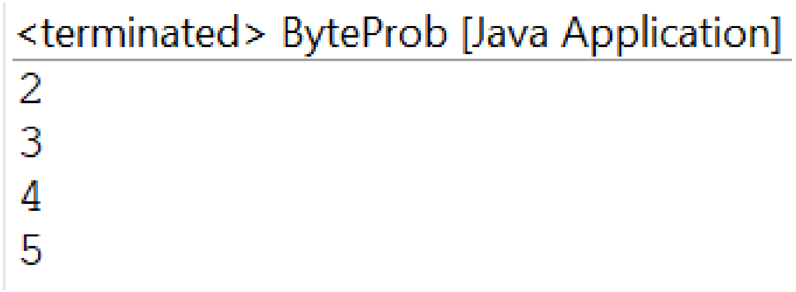
```
Sum of 90 and 40 is: -126
```



```java
public class ByteProb {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int x, y;
        x=Integer.MIN_VALUE;
        y=-x;

        if ((x != -y)) System.out.println("1");
        if(x == y) System.out.println("2");
        if ((x-y)==0) System.out.println("3");
        if ((x+y)==2*x) System.out.println("4");
        if ((byte)(-x)+x!=0) System.out.println("5");
    }
}
```



```java
public class ByteProb {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int x, y;
        x=Integer.MIN_VALUE;
        y=-x;

        if ((x != -y)) System.out.println("1");
        if(x == y) System.out.println("2");
        if ((x-y)==0) System.out.println("3");
        if ((x+y)==2*x) System.out.println("4");
        if ((byte)(-x)+x!=0) System.out.println("5")
    }
}
```



```
<terminated> ByteProb [Java Application]
2
3
4
5
```

From unsigned	To	Method
char	char	Preserve bit pattern; high-order bit becomes sign bit
char	short	Zero-extend
char	long	Zero-extend
char	unsigned short	Zero-extend
char	unsigned long	Zero-extend
short	char	Preserve low-order byte
short	short	Preserve bit pattern; high-order bit becomes sign bit
short	long	Zero-extend
short	unsigned char	Preserve low-order byte
long	char	Preserve low-order byte
long	short	Preserve low-order word
long	long	Preserve bit pattern; high-order bit becomes sign bit
long	unsigned char	Preserve low-order byte
long	unsigned short	Preserve low-order word

Key: Lost data   Misinterpreted data

Definitions

Secure = achieves some property despite attacks by adversaries.

- Systematic thought is required for successful defence.
- Details matter!
- High-level plan for thinking about security:

1. Goal: what your system is trying to achieve.
   - e.g. only Alice should read file F.
   - Common goals: confidentiality, integrity, availability.

High level plan

1. Policy: some plan (rules) that will get your system to achieve the goal.
   - e.g. set permissions on F so it's readable only by Alice's processes.
   - e.g. require a password and two-factor authentication.
2. Threat model: assumptions about what the attacker can do.
   - e.g. can guess passwords, cannot physically steal our server.
3. Mechanism: software/hardware that your system uses to enforce policy.
   - e.g. user accounts, passwords, file permissions, encryption.
   - policy might include human components (e.g., do not share passwords)
   - that's outside of the scope of the security mechanisms

Often layered: mechanism of one layer is policy of next level down.

High level plan

1. Policy: some plan (rules) that will get your system to achieve the goal.
   - e.g. set permissions on F so it's readable only by Alice's processes.
   - e.g. require a password and two-factor authentication.
2. Threat model: assumptions about what the attacker can do.
   - e.g. can guess passwords, cannot physically steal our server.
3. Mechanism: software/hardware that your system uses to enforce policy.
   - e.g. user accounts, passwords, file permissions, encryption.
   - policy might include human components (e.g., do not share passwords)
   - that's outside of the scope of the security mechanisms

Often layered: mechanism of one layer is policy of next level down.

Security Dimensions

- *Confidentiality*
  - Information in a system may be disclosed or made accessible to people or programs that are not authorized to have access to that information.
- *Integrity*
  - Information in a system may be damaged or corrupted making it unusual or unreliable.
- *Availability*
  - Access to a system or its data that is normally available may not be possible.

Why building secure system is hard?

- Example: SSD mark file, stored on an SCEBE SHAREPOINT.
  - Policy: only academics should be able to read and write the grades file.
  - Easy to implement the *positive* aspect of the policy:
  - allows academics to get at the file?
- But security is a **negative** goal:
  - We want no tricky way for a non- academics to get at the file.
- There are a huge number of potential attacks to consider!
  - Exploit a bug in the server's code.
  - Guess a academics' password.
  - Steal academics’ laptop, maybe it has a local copy of the grades file.
  - Intercept grades when they are sent over the network to the registrar.
  - Get a job in the registrar’s or school office.

Real-world scenarios

- One cannot get policies/threats/mechanisms right on the first try.
  - One must usually iterate:
  - Design, watch attacks, update understanding of threats and policies.
- Post-mortems important to understand
  - Public databases of vulnerabilities (e.g., https://cve.mitre.org/)
  - Encourage people to report vulnerabilities (e.g., bounty programs)
- Defender is often at a disadvantage in this game.
  - Defender usually has limited resources, other priorities.
  - Defender must balance security against convenience.
- A determined attacker can usually win!
- Defence in depth
- Recovery plan (e.g., secure backups)
  - Most of this lecture is about failures to make you start thinking in this way

Perfect security is rarely achieved

- What's the point if we can't achieve perfect security?
  - Perfect security is rarely required.
- Make cost of attack greater than the value of the information.
  - So that perfect defences aren't needed.
- Make our systems less attractive than other peoples'.
  - Works well if attacker e.g. just wants to generate spam.
- Find techniques that have big security payoff (i.e. not merely patching holes).
  - Successful: popular attacks from 10 years ago are no longer very fruitful.
  - Sometimes security **increases** value for defender:
  - VPNs might give employees more flexibility to work at home.
  - Sandboxing (JavaScript, Native Client) might give me more confidence to run software I don't fully understand.
  - No perfect physical security either. But that's OK: cost, deterrence.
- One big difference in computer security: attacks are cheap.

Problems with Policy

- I.e. system correctly enforces policy -- but policy is inadequate.

Example 1: Business-class airfare.

  - Airlines allow business-class tickets to be changed at any time, no fees.
  - Is this a good policy?
  - Turns out, in some systems ticket could have been changed even AFTER boarding.
  - Adversary can keep boarding plane, changing ticket to next flight, ad infinitum.

- Revised policy: ticket cannot be changed once passenger has boarded the flight.
- Sometimes requires changes to the system architecture.
  - Need computer at the aircraft gate to send updates to the reservation system.

Problems with Policy

Policy Flawed Example 2: Fairfax County, VA school system.

- Student can access only his/her own files in the school system.
- Superintendent has access to everyone's files.
- Teachers can add new students to their class.
- Teachers can change password of students in their class.
  - What's the worst that could happen if student gets teacher's password?
- Student adds the superintendent to the compromised teacher's class.
  - Changes the superintendent's password, since they're a student in class.
  - Logs in as superintendent and gets access to all files.
- Policy amounts to: teachers can do anything.

Ref: http://catless.ncl.ac.uk/Risks/26.02.html#subj7.1

Problems with Policy

Policy Flawed Example 3: Mat Honan an editor at wired.com;

his accounts at Amazon, Apple, Google, etc.

- Gmail password reset: send a verification link to a backup email address.
- Google helpfully prints part of the backup email address.
- Mat Honan's backup address was his Apple @me.com account.
- Apple password reset: need billing address, last 4 digits of credit card.
- Address is easy, but how to get the 4 digits?
- How to get hold of that e-mail?
- Can use the credit card just added to the account.
- Now go to Amazon's web site and request a password reset.
- Reset link sent to the new e-mail address.
- Now log in to Amazon account, view saved credit cards.
- Amazon doesn't show full number but DOES show last 4 digits of all cards.

Problems with Policy

Policy Flawed Example 3: Mat Honan an editor at wired.com;his accounts at Amazon, Apple, Google, etc. (CONTINUE)

  - Amazon doesn't show full number but DOES show last 4 digits of all cards.
  - Including the account owner's original cards!
  - Now attacker can reset Apple password, read Gmail reset e-mail,
  - reset Gmail password.

- Lesson: attacks often assemble apparently unrelated trivia.
- Lesson: individual policies OK, but combination is not.
  - Apple views last 4 as a secret, but many other sites do not.
- Lesson: big sites cannot hope to identify which human they are talking to; at best "same person who originally created this account".
- security questions and e-mailed reset link are examples of this.

https://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/all/

Example 4: Insecure defaults.

- Well-known default passwords in routers.
- Public default permissions in cloud services (e.g., objects in AWS S3 bucket).
- Secure defaults are crucial because of the "negative goal" aspect.
- Large systems are complicated, lots of components.
- Operator might forget to configure some component in their overall system.
- Important for components to be secure if operator forgets to configure them.

Reason policies go wrong

Policies typically go wrong in "management" or "maintenance" cases.

- Who can change permissions or passwords?
- Who can access audit logs?
- Who can access the backups?
- Who can upgrade the software or change the configuration?
- Who can manage the servers?
- Who revokes privileges of former admins / users / ...?

Flawed threat models / assumptions

  - I.e. designer assumed an attack wasn't feasible (or didn't think of the attack).
- Example: assume the design/implementation is secret
  - "Security through obscurity“,   Clipper chip
  - https://en.wikipedia.org/wiki/Clipper_chip
- Broken secret crypto functions
  - In 1994, Matt Blaze identified a vulnerability in the Clipper chip's escrow system in issue involved a 128-bit "Law Enforcement Access Field" (LEAF).
  - The short hash length made the system vulnerable to brute-force attacks, allowing the Clipper chip to encrypt communications while disabling its key escrow function.
  - In 1995, Yair Frankel and Moti Yung presented another inherent design flaw, demonstrating that the LEAF of one device could be attached to messages from another, circumventing the escrow system in real time.
  - In 1997, a group of leading cryptographers published a paper highlighting the architectural weaknesses of key escrow systems, including the Clipper chip's Skipjack protocol, emphasizing the risks involved in such designs.

Flawed threat models / assumptions

- Example: most users are not thinking about security.
- User gets e-mail saying, "click here to renew your account", then plausible-looking page asks for their password.
- Or dialog box pops up with "Do you really want to install this program?"
- Or tech support gets call from convincing-sounding user to reset password.

Example 1: computational assumptions change over time

- MIT's Kerberos system used 56-bit DES keys, since mid-1980's.
  - At the time, seemed fine to assume adversary can't check all 2^56 keys.
  - No longer reasonable: now costs about $100.

- Several years ago, final project in some universities showed can get any key in a day.

- https://www.cloudcracker.com/dictionaries.html

Example 2: computational assumptions change over time.

- Example: assuming a particular kind of a solution to the problem.
  - Many services use CAPTCHAs to check if a human is registering for an account.
  - Requires decoding an image of some garbled text, for instance.
- Goal is to prevent mass registration of accounts to limit spam, prevent
  - high rate of password guessing, etc.
- Assumed adversary would try to build OCR to solve the puzzles.
  - Good plan because it's easy to change image to break the OCR algorithm.
  - Costly for adversary to develop new OCR!
- Turns out adversaries found another way to solve the same problem.
  - Human CAPTCHA solvers in third-world countries.
  - Human solvers are far better at solving CAPTCHAs than OCRs or even regular users.
  - Cost is very low (fraction of a cent per CAPTCHA solved).
- https://www.cs.uic.edu/pub/Kanich/Publications/re.captchas.pdf

Example: assuming you are running the expected software.

1. In the 80's, military encouraged research into secure OS'es.
   - Surprise: successful attacks by gaining access to development systems
   - Mistake: implicit trust in compiler, developers, distribution, &c
2. Apple's development tools for iPhone applications (Xcode) are large.
   - Downloading them from China required going to Apple's servers outside of China.
   - Takes a long time.
   - Unofficial mirrors of Xcode tools inside China.
   - Some of these mirrors contained a modified version of Xcode that injected malware into the resulting iOS applications.
   - Found in a number of high-profile, popular iOS apps!

https://en.wikipedia.org/wiki/XcodeGhost

Problems with the mechanism - bugs

Bugs routinely undermine security.

- Rule of thumb: one bug per 1000 lines of code.
- Bugs in implementation of security policy.
- But also bugs in code that may seem unrelated to security, but they are not.

- ***Good mindset: Any bug is a potential security exploit.***
  - Especially if there is no isolation around the bug.

Apple's iCloud password-guessing

- People often pick weak passwords; can often guess w/ few attempts (1K-1M).
- Most services, including Apple's iCloud, rate-limit login attempts.
- Apple's iCloud service has many APIs.
- One API (the "Find my iPhone" service) forgot to implement rate-limiting.
- Attacker could use that API for millions of guesses/day.
- Lesson: if many checks are required, one will be missing.

https://github.com/hackappcom/ibrute

Missing access control checks in Citigroup's credit card web site.

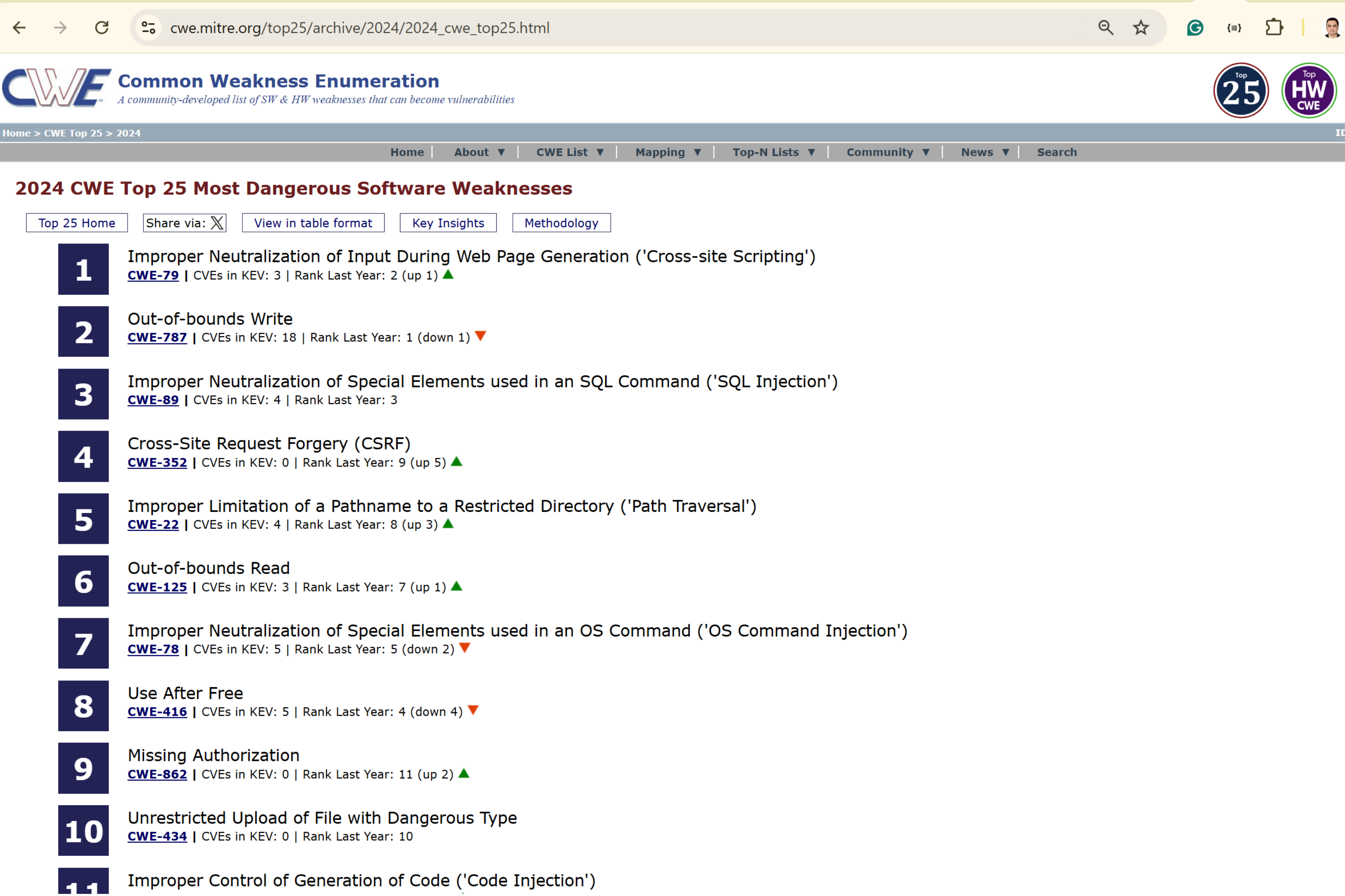
- Citigroup allowed credit card users to access their accounts online.
- Login page asks for username and password.
- If username and password OK, redirected to account info page.
- The URL of the account info page included some numbers.
  - e.g. x.citi.com/id=1234
- The numbers were (related to) the user's account number.
- Adversary tried different numbers, got different people's account info.
- The server didn't check that you were logged into that account!

- Lesson: programmers tend to think only of intended operation.

http://www.nytimes.com/2011/06/14/technology/14security.html

References for vulnerabilities

- **CWE™** is a community-developed list of software and hardware weakness types. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.
  - Top 25 CWEs: https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html
  - Top 10 KVE: CWE - 2023 CWE Top 10 KEV Weaknesses (mitre.org)
- **CVE:** The mission of the CVE® Program is to identify, define, and catalogue publicly disclosed cybersecurity vulnerabilities.
  - CWE - 2023 CWE Top 10 KEV Weaknesses (mitre.org)



cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html
Top 25
Top HW CWE

Common Weakness Enumeration

A community-developed list of SW & HW weaknesses that can become vulnerabilities

Home > CWE Top 25 > 2024
Home | About ▼ | CWE List ▼ | Mapping ▼ | Top-N Lists ▼ | Community ▼ | News ▼ | Search

2024 CWE Top 25 Most Dangerous Software Weaknesses

Top 25 Home
Share via: X
View in table format
Key Insights
Methodology

1 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-79 | CVEs in KEV: 3 | Rank Last Year: 2 (up 1) ▲

2 Out-of-bounds Write
CWE-787 | CVEs in KEV: 18 | Rank Last Year: 1 (down 1) ▼

3 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-89 | CVEs in KEV: 4 | Rank Last Year: 3

4 Cross-Site Request Forgery (CSRF)
CWE-352 | CVEs in KEV: 0 | Rank Last Year: 9 (up 5) ▲

5 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-22 | CVEs in KEV: 4 | Rank Last Year: 8 (up 3) ▲

6 Out-of-bounds Read
CWE-125 | CVEs in KEV: 3 | Rank Last Year: 7 (up 1) ▲

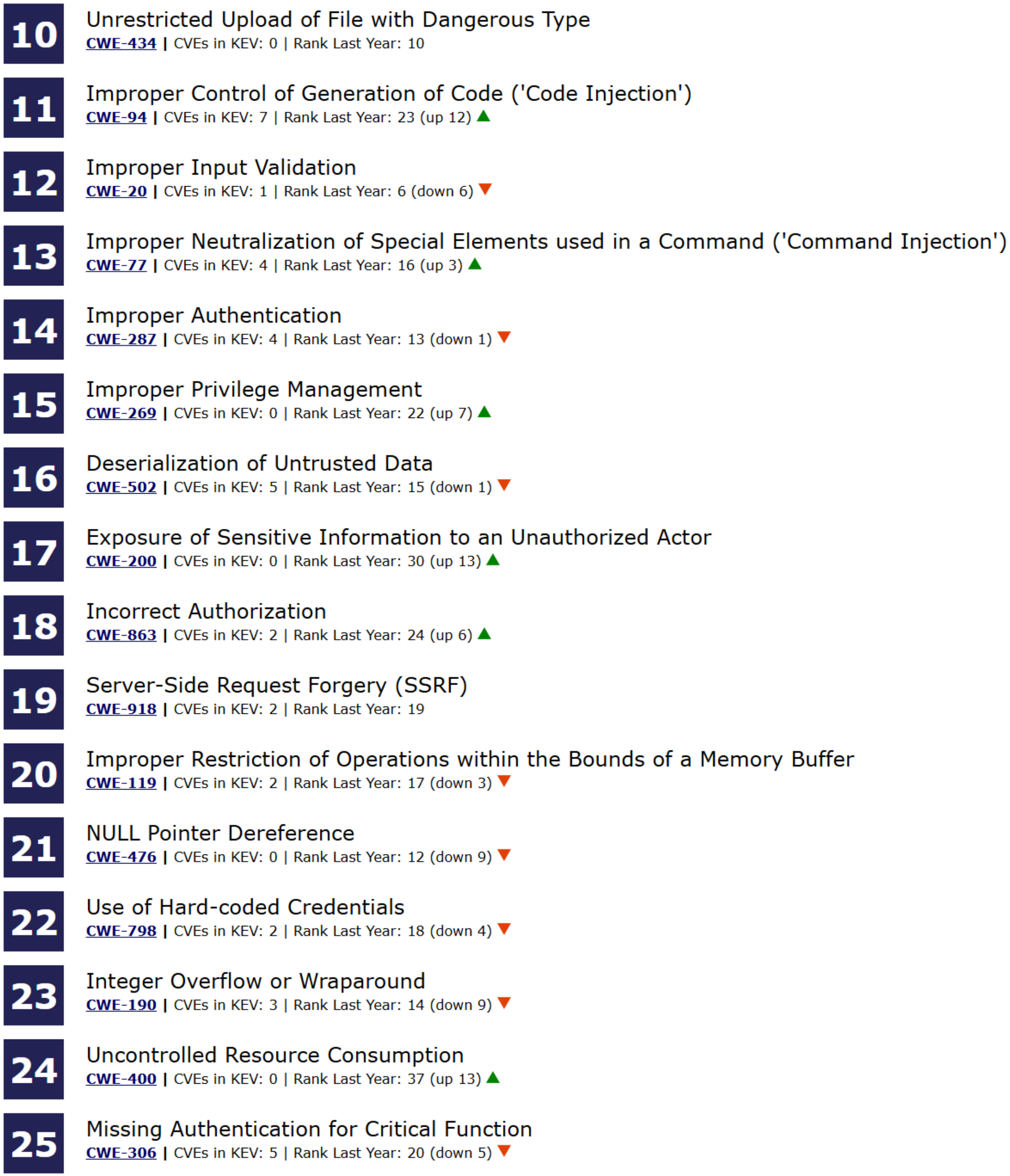
7 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-78 | CVEs in KEV: 5 | Rank Last Year: 5 (down 2) ▼

8 Use After Free
CWE-416 | CVEs in KEV: 5 | Rank Last Year: 4 (down 4) ▼

9 Missing Authorization
CWE-862 | CVEs in KEV: 0 | Rank Last Year: 11 (up 2) ▲

10 Unrestricted Upload of File with Dangerous Type
CWE-434 | CVEs in KEV: 0 | Rank Last Year: 10

11 Improper Control of Generation of Code ('Code Injection')



10 Unrestricted Upload of File with Dangerous Type
CWE-434 | CVEs in KEV: 0 | Rank Last Year: 10

11 Improper Control of Generation of Code ('Code Injection')
CWE-94 | CVEs in KEV: 7 | Rank Last Year: 23 (up 12) ▲

12 Improper Input Validation
CWE-20 | CVEs in KEV: 1 | Rank Last Year: 6 (down 6) ▼

13 Improper Neutralization of Special Elements used in a Command ('Command Injection')
CWE-77 | CVEs in KEV: 4 | Rank Last Year: 16 (up 3) ▲

14 Improper Authentication
CWE-287 | CVEs in KEV: 4 | Rank Last Year: 13 (down 1) ▼

15 Improper Privilege Management
CWE-269 | CVEs in KEV: 0 | Rank Last Year: 22 (up 7) ▲

16 Deserialization of Untrusted Data
CWE-502 | CVEs in KEV: 5 | Rank Last Year: 15 (down 1) ▼

17 Exposure of Sensitive Information to an Unauthorized Actor
CWE-200 | CVEs in KEV: 0 | Rank Last Year: 30 (up 13) ▲

18 Incorrect Authorization
CWE-863 | CVEs in KEV: 2 | Rank Last Year: 24 (up 6) ▲

19 Server-Side Request Forgery (SSRF)
CWE-918 | CVEs in KEV: 2 | Rank Last Year: 19

20 Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-119 | CVEs in KEV: 2 | Rank Last Year: 17 (down 3) ▼

21 NULL Pointer Dereference
CWE-476 | CVEs in KEV: 0 | Rank Last Year: 12 (down 9) ▼

22 Use of Hard-coded Credentials
CWE-798 | CVEs in KEV: 2 | Rank Last Year: 18 (down 4) ▼

23 Integer Overflow or Wraparound
CWE-190 | CVEs in KEV: 3 | Rank Last Year: 14 (down 9) ▼

24 Uncontrolled Resource Consumption
CWE-400 | CVEs in KEV: 0 | Rank Last Year: 37 (up 13) ▲

25 Missing Authentication for Critical Function
CWE-306 | CVEs in KEV: 5 | Rank Last Year: 20 (down 5) ▼

Edinburgh Napier
UNIVERSITY

CWE-798: Use of Hard-coded Credentials

- The product contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

CWE-798: Use of Hard-coded Credentials

Description

- Hard-coded credentials typically create a significant hole that allows an attacker to bypass the authentication that has been configured by the product administrator.

- This hole might be difficult for the system administrator to detect. Even if detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely.

CWE-798: Use of Hard-coded Credentials
Main Variations: Inbound

- The product contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials.
  - vulnerability CVE-2017-14143: Kaltura server before 13.2.0 contained a code that allowed access to the web application to any user with pre-set "userzone" cookie equal to "y3tAno3therS$cr3T".

CWE-798: Use of Hard-coded Credentials

Main Variations: Inbound

- This hard-coded password is the same for each installation of the product, and cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the product.
- If the password is ever discovered or published, then anybody with knowledge of this password can access the product.
- Finally, since all installations of the product will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

CWE-798: Use of Hard-coded Credentials

Main Variations: Outbound

- The product connects to another system or component, and it contains hard-coded credentials for connecting to that component.
- The Outbound variant applies to front-end systems that authenticate with a back-end service.

CWE-798: Use of Hard-coded Credentials

Main Variations: Outbound

- The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end product. Any user of that program may be able to extract the password.
- Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

CWE-798: Use of Hard-coded Credentials
Example 1

- The following code uses a hard-coded password to connect to a database:

Language: Java

```
...
DriverManager.getConnection(url, "scott", "tiger");

...
```

CWE-798: Use of Hard-coded Credentials

Example 2

- The following code is an example of an internal hard-coded password in the back-end:

Language: Java

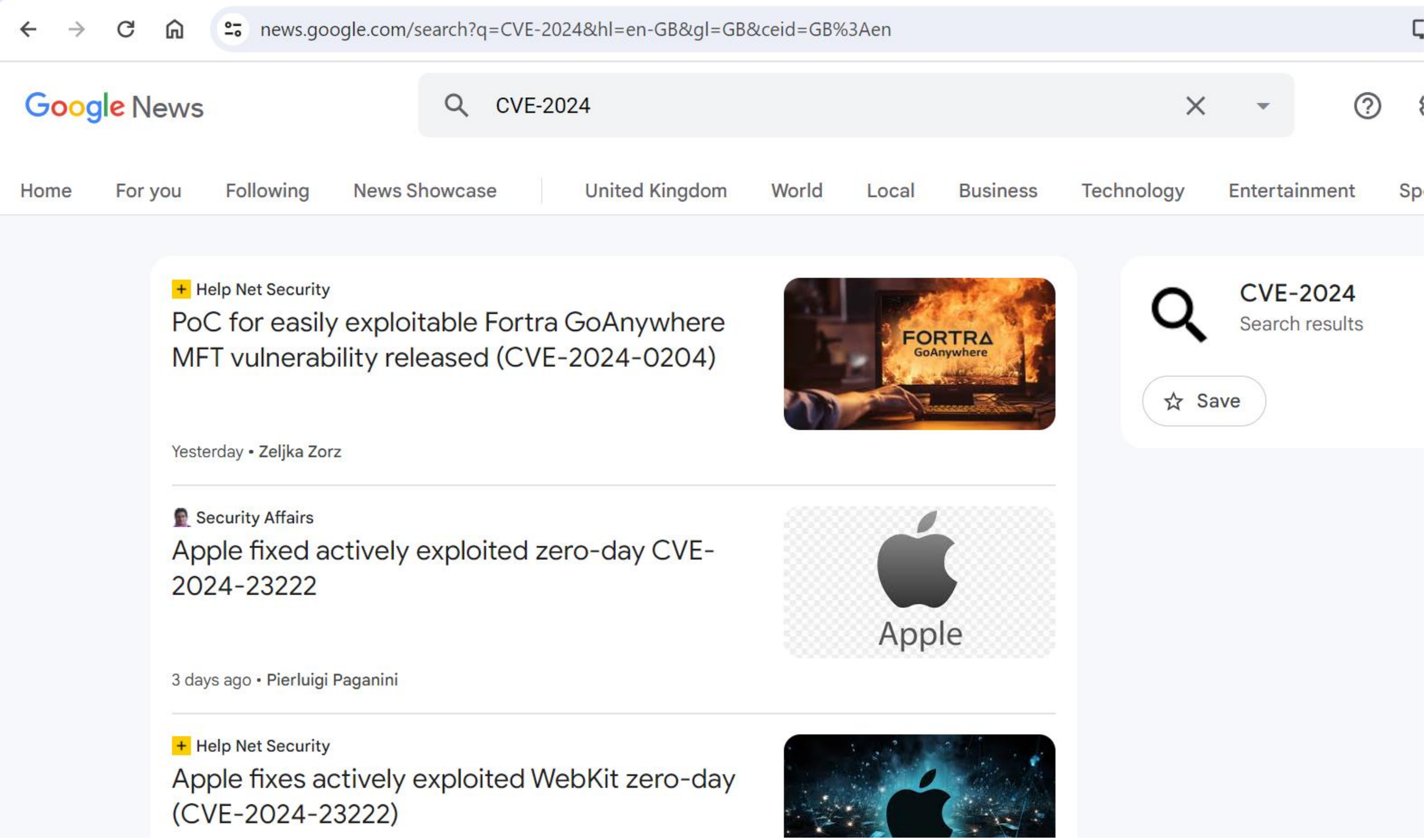
```
int VerifyAdmin(String password) {
        if (!password.equals("Mew!")) {
        return(0)
        }
        //Diagnostic Mode
        return(1);
        }
```

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

- The product uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory,
- But the product does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.

References for vulnerabilities

- **NVD:** The NVD is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP).
- This data enables automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklist references, security-related software flaws, product names, and impact metrics.
  - https://nvd.nist.gov/



news.google.com/search?q=CVE-2024&hl=en-GB&gl=GB&ceid=GB%3Aen

CVE-2024

Help Net Security

PoC for easily exploitable Fortra GoAnywhere MFT vulnerability released (CVE-2024-0204)

Yesterday • Zeljka Zorz

Security Affairs

Apple fixed actively exploited zero-day CVE-2024-23222

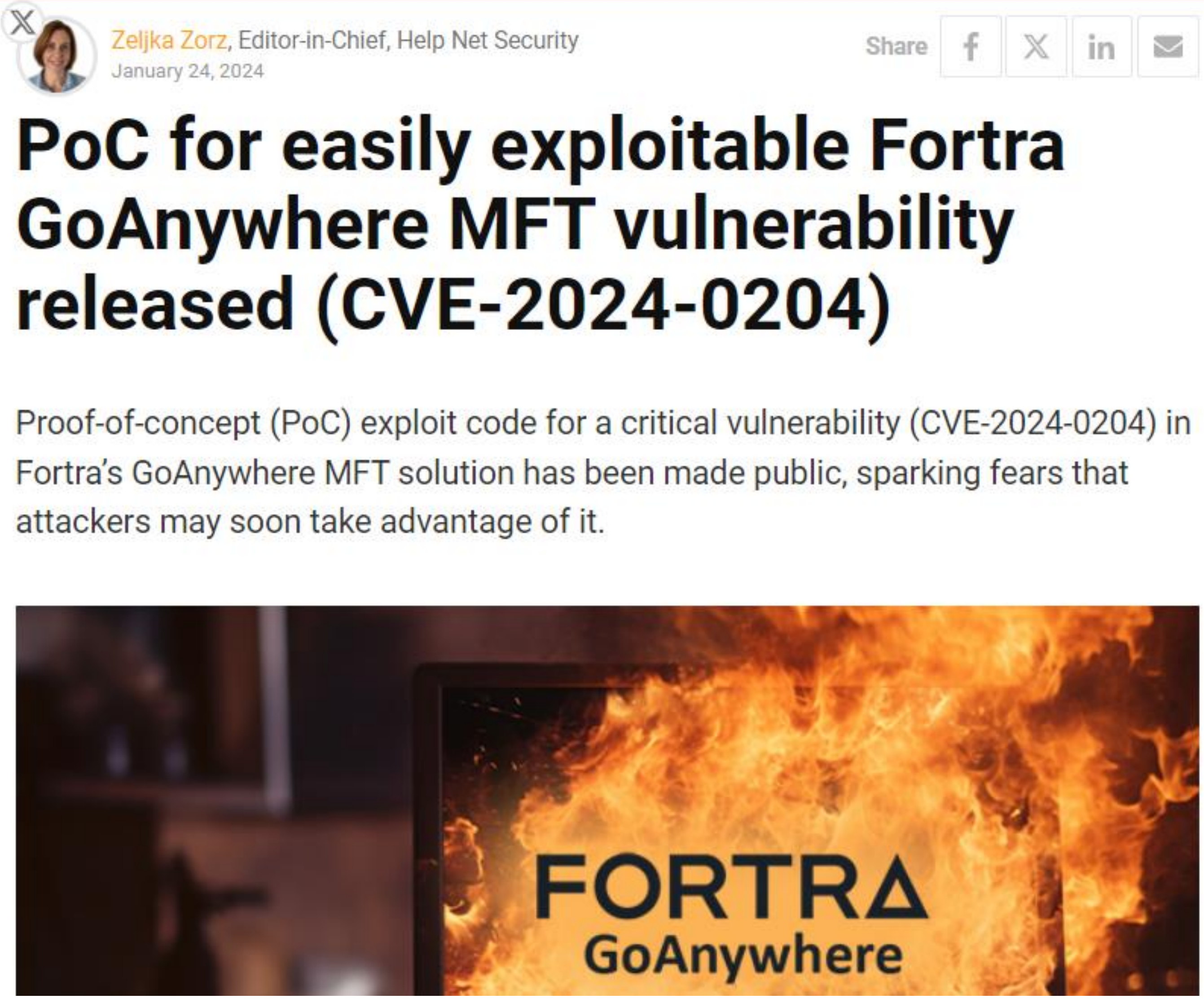
3 days ago • Pierluigi Paganini

Help Net Security

Apple fixes actively exploited WebKit zero-day (CVE-2024-23222)

CVE-2024

Search results



Zeljka Zorz, Editor-in-Chief, Help Net Security
January 24, 2024

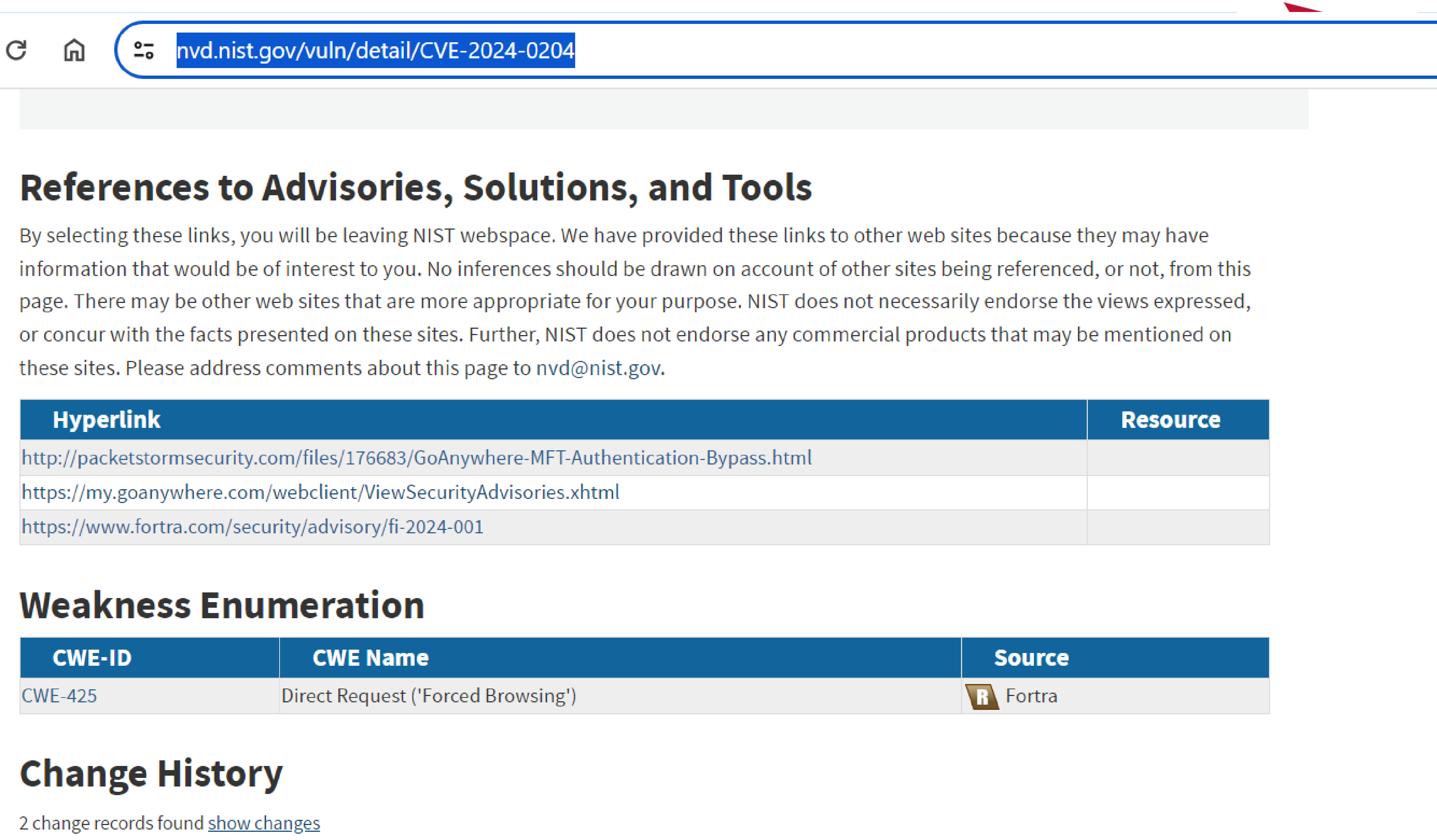
PoC for easily exploitable Fortra GoAnywhere MFT vulnerability released (CVE-2024-0204)

Proof-of-concept (PoC) exploit code for a critical vulnerability (CVE-2024-0204) in Fortra's GoAnywhere MFT solution has been made public, sparking fears that attackers may soon take advantage of it.

CVE-2024-0204 in Fortra's GoAnywhere MFT software

- CVE-2024-0204 is a critical authentication bypass vulnerability in Fortra's GoAnywhere Managed File Transfer (MFT) software, affecting versions 6.x (from 6.0.1) and 7.x (up to 7.4.1).
- It allows unauthorized creation of admin users via the admin portal. Addressed in version 7.4.1, released on December 7, 2023.
- Fortra advised customers to upgrade or apply specific workarounds for non-container and container deployments.
- A Proof of Concept (PoC) was published by Horizon3.ai.
- Over 1,800 internet-exposed instances are potentially at risk, and the vulnerability's simplicity raises significant concerns for easy exploitation and potential extortion.

- *However, it is not mentioned what CWE it belongs to*



nvd.nist.gov/vuln/detail/CVE-2024-0204

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
http://packetstormsecurity.com/files/176683/GoAnywhere-MFT-Authentication-Bypass.html	
https://my.goanywhere.com/webclient/ViewSecurityAdvisories.xhtml	
https://www.fortra.com/security/advisory/fi-2024-001	

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-425	Direct Request ('Forced Browsing')	R Fortra

Change History

2 change records found show changes

https://nvd.nist.gov/vuln/detail/CVE-2024-0204

QUESTIONS AND COMMENTS

- To contact me: a.sami@napier.ac.uk

Disclaimer

- Slides for this section were based on contents of MIT module 6.858 Computer Systems Security 2022
- CWE's are based on Top 25 most dangerous vulnerabilities in 2024: https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html