Debasis
Mohanty

Software Security Engineering

Learnings from the past to fix the future

OWASP's 20th Anniversary Event
24-25 September 2021

Debasis Mohanty

Head of Technical Services
SEQA Security
www.seqa.co.nz

Over 20 years of experience in doing offensive and defensive security

20th ANNIVERSARY OWASP

Who am I?

How my experience is relevant to this talk?

SEQA™ Information Security

- Head of Security Services at SEQA Security (a New Zealand based company)
- Over 20 years of Offensive and Defensive Security Experience (since 1997-1998)
  - The vast majority of the experience has been vulnerability research-focused and exploit development
  - Over 10+ years of Software Security Engineering Background
  - Led Security Engineering CoE of mid-sized and large Technology Companies
  - Worked closely with the multiple engineering teams to integrate security across SDLC
- A **simple security guy** who **likes to solve complex security problems** using simple methods

Personal Website: coffeeandsecurity.com Twitter: @coffeensecurity Email: d3basis.m0hanty@gmail.com

This talk is broken down into four parts

Overview

- **The History:**
  Historical data shows we continue to see around two decades old security bugs
- **The Reason:**
  Why do we still continue to see one to two decades old security bugs?
- **The Solution:**
  The top two mitigation strategies to consider based on the past learnings
- **The Misconception:**
  The Silver Bullet In Software Security Engineering

Let's begin with the history and look at the State of Software Security Vulnerabilities

The History:

The Present State of Security Vulnerabilities:

Historical data shows we continue to see around two decades old security bugs.

Top Application Security Vulnerabilities

That has be around for over two decades

- Cross Site Scripting (webapp)

  As per Wikipedia: https://en.wikipedia.org/wiki/Cross-site_scripting

  - Microsoft security-engineers introduced the term "cross-site scripting" in January 2000
  - XSS vulnerabilities have been reported and exploited since the 1990s

- SQL Injection (webapp and OS-native apps)

  As per Wikipedia: https://en.wikipedia.org/wiki/SQL_injection

  - The first public discussions of SQL injection started appearing around 1998 (an article in Phrack Magazine)

- Deserialization Issue (web-app, OS-native apps)
  - 01 Aug 2002: Integer overflow in xdr_array() function when deserializing the XDR stream
    https://www.kb.cert.org/vuls/id/192995

Top OS and OS-Native Apps Vulnerabilities

That has be around for over one to two decades

- Buffer Overflow

  As per Wikipedia: https://en.wikipedia.org/wiki/Buffer_overflow
  - Buffer overflows were understood and partially publicly documented as early as 1972
  - The earliest documented hostile exploitation of a buffer overflow was in 1988 (Morris worm)
  - In 1996: Phrack magazine article "Smashing the Stack for Fun and Profit“ by Elias Levy (aka Aleph One)
- Race Condition (OS, OS-Native apps and webapps)
  - May 1995: Publication title “A Taxonomy of UNIX System and Network Vulnerabilities” https://cwe.mitre.org/documents/sources/ATaxonomyofUnixSystemandNetworkVulnerabilities%5BBishop95%5D.pdf
  - CVE-2001-0317: https://nvd.nist.gov/vuln/detail/CVE-2001-0317
- Use-After-Free (UAF) and Double Free
  - CVE-2006-4997: Freed pointer dereference in the clip_mkip function in net/atm/clip.c of the ATM subsystem in Linux kernel
  - CVE-2002-0059: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0059
  - More examples of double free: https://cwe.mitre.org/data/definitions/415.html

History of Few Common Bug Classes

- Observations:
  - The majority of the bug classes in the list have been around two decades
  - This list relates to bugs affecting multiple applications and software.
  - The count of bugs across each year may not necessarily be accurate.
  - However, you get an idea that these bugs have been around for a long period
- Conclusion:

  Given that these bug classes have been around for two decades, it implies that something is not right with how the Industry has dealt with these bugs.

cvedetails.com/vulnerabilities-by-types.php

Screenshot date: 15 September 2021

Vulnerabilities By Type

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
1999	894	177	112	172			2	7		25	16	103			2
2000	1020	257	208	206		2	4	20		48	19	139			
2001	1677	403	403	297		7	34	124		83	36	220		2	2
2002	2156	498	553	435	2	41	200	103		127	76	199	2	14	1
2003	1527	381	477	372	2	50	129	60	1	62	69	144		16	5
2004	2451	580	614	408	3	148	291	111	12	145	96	134	5	38	5
2005	4935	838	1627	657	21	604	786	202	15	289	261	221	11	100	14
2006	6610	893	2719	664	91	967	1302	322	8	267	272	184	18	849	30
2007	6520	1101	2601	955	95	706	883	338	14	267	326	242	69	700	45
2008	5632	894	2310	699	128	1101	807	362	7	288	268	188	83	170	76
2009	5736	1035	2185	698	188	963	851	323	9	337	302	223	115	138	738
2010	4653	1102	1714	676	342	520	605	276	8	234	284	238	86	73	1501
2011	4155	1221	1334	734	351	294	470	108	7	197	411	206	58	17	557
2012	5297	1425	1459	833	423	243	759	122	13	344	392	250	166	14	623
2013	5191	1455	1186	856	366	156	650	110	7	352	512	274	123	1	206
2014	7939	1599	1572	841	420	304	1103	204	12	457	2106	239	264	2	403
2015	6504	1793	1830	1084	749	221	784	151	12	577	753	366	248	5	129
2016	6454	2029	1496	1312	717	94	498	99	15	444	870	602	86	7	1
2017	14714	3155	3004	2494	745	508	1518	279	11	629	1659	459	327	18	6
2018	16557	1853	3041	2121	400	517	2048	545	11	708	1238	247	461	31	4
2019	17344	1342	3201	1286	488	549	2390	465	10	710	981	202	535	57	13
2020	18325	1351	3248	1604	409	460	2178	401	14	966	1338	310	402	37	62
2021	13759	1312	2742	1156	310	458	1794	314	3	572	622	186	293	28	
Total	160050	26694	39636	20560	6250	8913	20086	5046	189	8128	12907	5576	3352	2317	4423

Note: This may not be the most comprehensive list but you get the overall picture.

The Big Question

So, why do we continue to see one to two decades-old security bugs?

The Reason(s)

There are many reasons, but here we will discuss the two most prominent reasons.

The most common reason: This bug is not my problem; it is someone else's problem.

The Two Most Prominent Reasons

The Reasons

The two most prominent reasons are obscured within the way the vast majority of the Organisation responds to a bug report of the applications and software:

- They **are** responsible for supporting
- They **aren't** responsible for supporting

Note: While there are many reasons but here we will discuss the two most prominent reasons

Reason No. 1

Typical Response For A Bug Report

(of the applications and software you support)

The Reason No. 1

Typical vulnerability mitigation strategy, upon receiving a bug report affecting the software you are responsible for:

- Fix exactly what is reported
- Fix exactly what is reported including any other instances of the same bug
- Fix based on the bugs risk rating but follow the second approach

While this is fine but…

Disadvantage of Such Mitigation Strategies

Common Mitigation Strategies	Disadvantages
You fix a reported bug but do not check for any bug instances or variants in the same application.	You are likely to miss other instances and variants of the same bug in the application (if they exist).
You fix all instances and variants of a particular bug in an application but do not check whether similar bugs exist in other applications you support.	You are likely to miss instances and variants of the same bug if they exist in other applications.
You follow the second approach but fix issues with relatively higher risk ratings (e.g. critical/high/medium) but do not fix any lower risk rating issue.	Several historical evidence shows that bugs that look low hanging or trivial can be combined with other bugs to perform a more practical attack.

If such mitigation strategies resonate with your bug mitigation practices, you are far from making your application and software resilient against known security bugs.

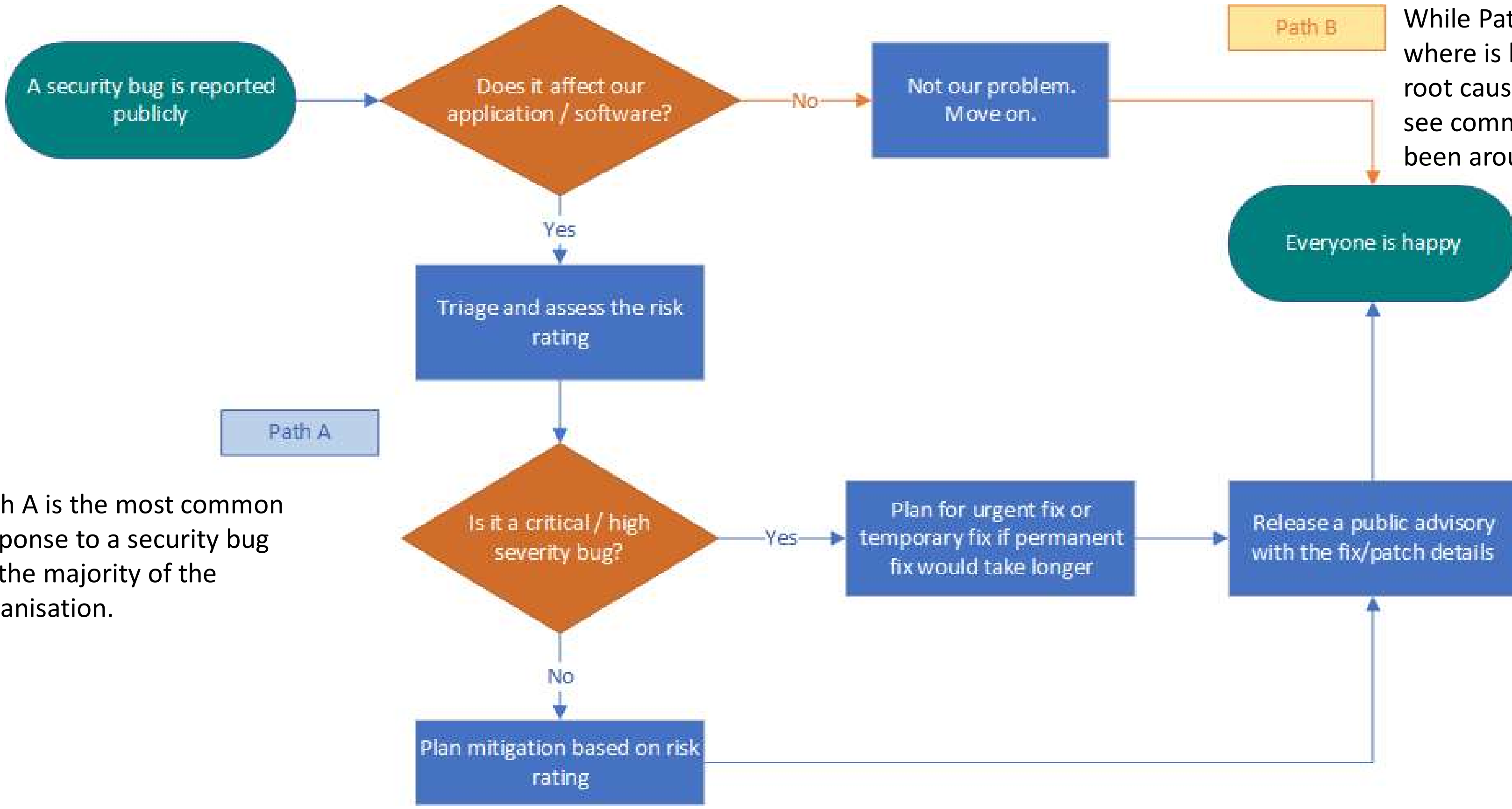
Reason No. 2

The Way "The Industry" Respond

To Any Publicly Reported Security Bug

Reason No.2

While Path B may look obvious, this is where is hidden one of the significant root causes of why we continue to see common security bugs that have been around for over 1-2 decades.



Path A is the most common response to a security bug by the majority of the organisation.

The flow chart illustrates the most common approach across the industry while dealing with or responding to a bug report.

The Solution

Tackling security vulnerabilities going forward based on the learnings from the past

No.1 - Learnings from the past

Learnings from the historical records of all the known bugs

Let's start by understanding the difference between a Bug Class and Bug Nature.

Understanding Bug Class and Bug Nature

- Class of the bug can be described as **the way a particular bug is exploited and/or it's resulting impact**.

- Nature of the bug primarily relates to **the root cause of the bug**.

  - Example 1: Cross-Site Scripting in a file upload page
    - Here the bug class is Cross Site Scripting.
    - However, the nature of the bug is 'missing sanitisation of tainted inputs'

  - Example2: SQL Injection in an authentication form
    - SQL Injection is a bug class name.
    - However, the nature of bug is insecure interpretation of tainted inputs as commands.

The corresponding root cause and bug nature of a bug class.

Translating A Bug Class

To It's Corresponding Root Cause and Bug Nature

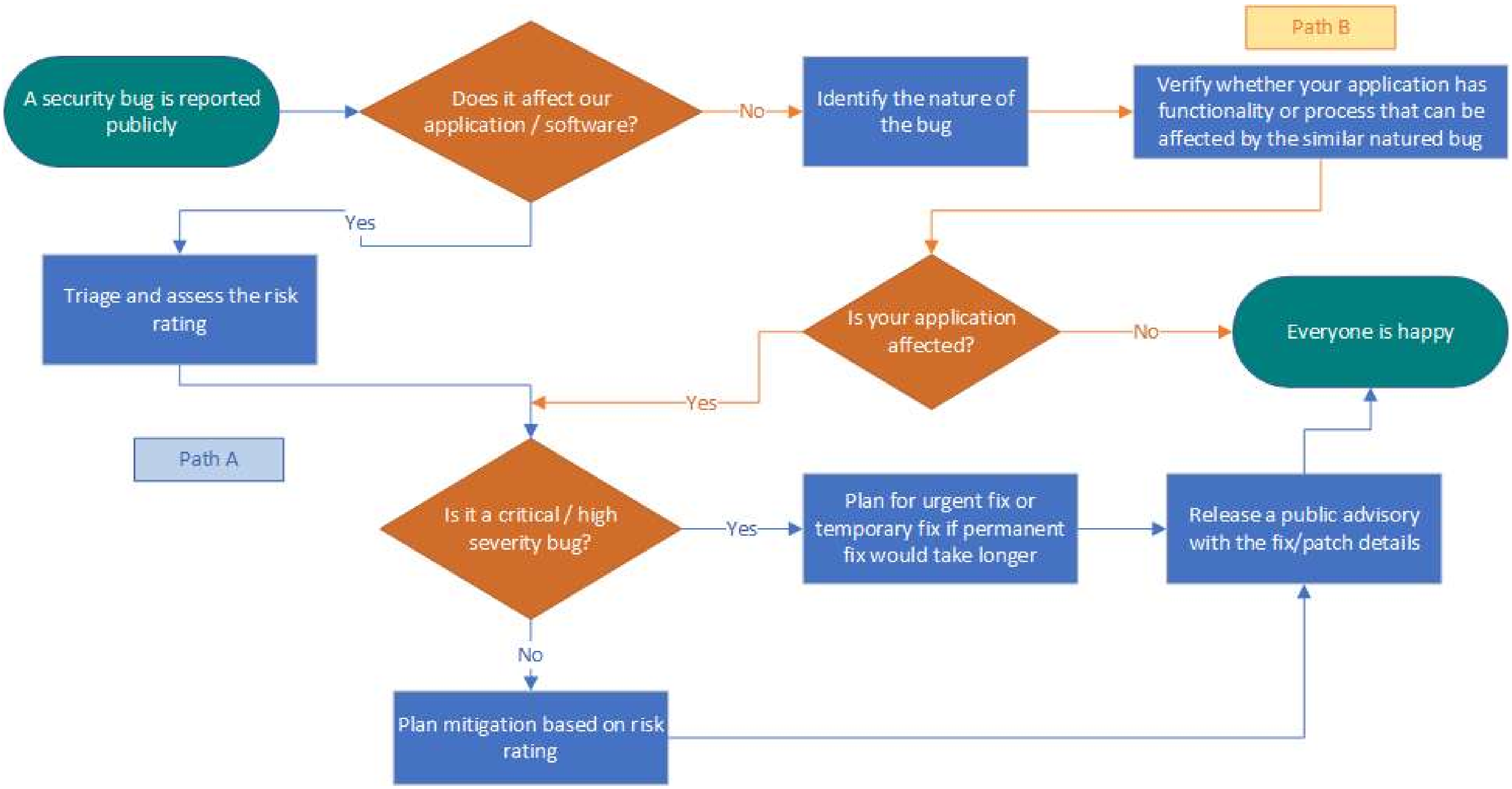
Bug Class / Type	Root Cause	Bug Nature
Cross Site Scripting	When the tainted input becomes output without sanitisation	• Injection Flaw
SQL Injection	When tainted input becomes command	• Injection Flaw • Insecure Interpretation of Input
Cross Site Request Forgery	Lack of server-side mechanism to differentiate between legit and forged request	• Trust Boundary Violation
Broken Access Control	Missing or inadequate check against required permissions	• Trust Boundary Violation • Inadequate Session Management
Command Injection	When tainted input becomes command	• Injection Flaw • Insecure Interpretation of Input

Note: The above list is not comprehensive. Instead, these are few examples provided as a guideline to understand the difference between a bug class and the bug nature or the root cause.

The approach towards mitigating and managing security vulnerabilities

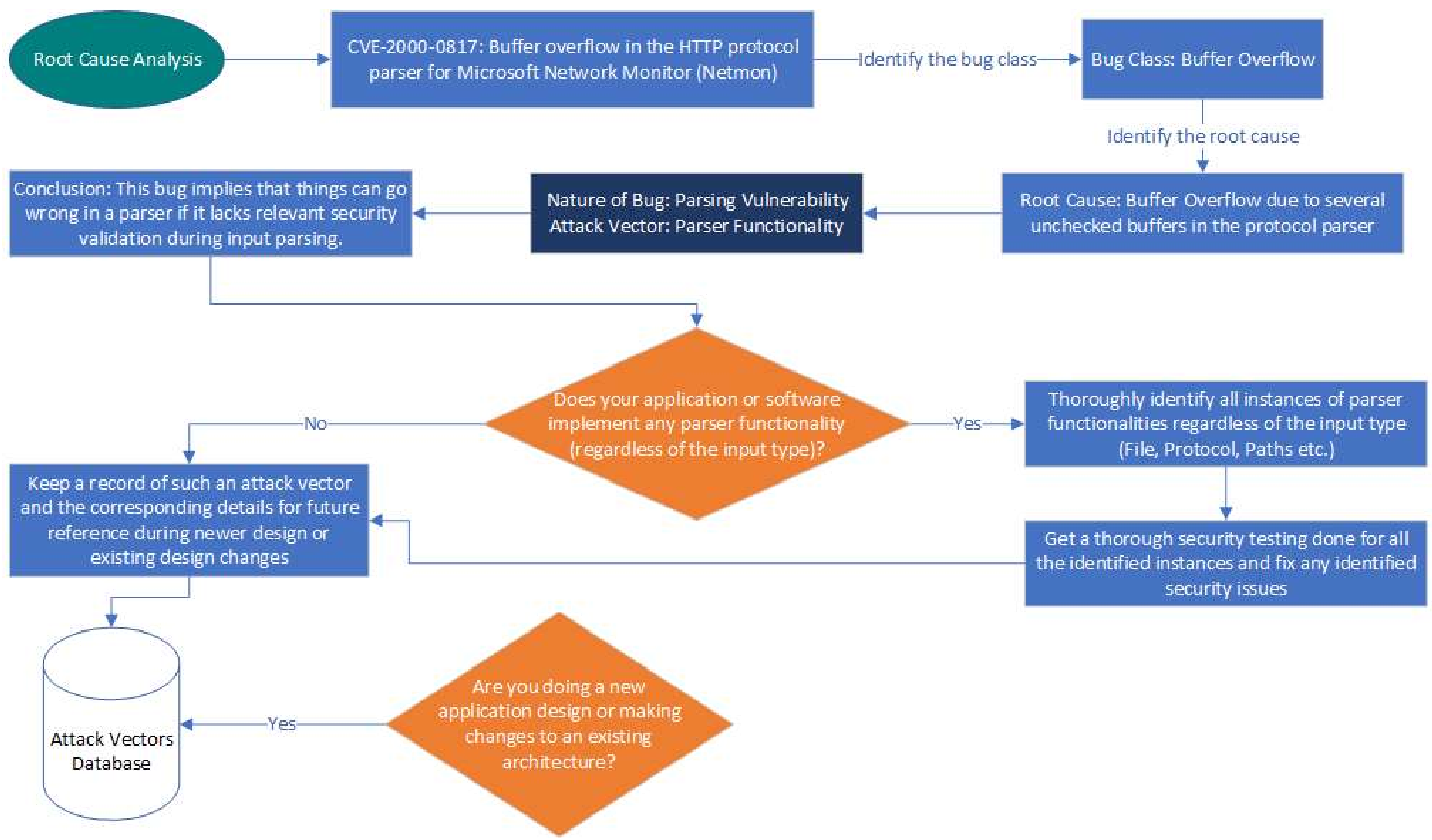
The Way “The Industry” Must Respond

To Any Publicly Reported Bugs



Decoding The Nature of a Bug (MS00-083)

CVE-2000-0817 (MS00-083): Buffer overflow in the HTTP protocol parser for Microsoft Network Monitor (Netmon) allows remote attackers to execute arbitrary commands via malformed data, aka the "Netmon Protocol Parsing" vulnerability.



Decoding The Nature of a Bug

(More Examples)

- File **Parsing** Vulnerabilities
  - MS04-007: ASN.1 parsing vulnerability (828028)
  - MS04-028: Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution
- Protocol **Parsing** Vulnerabilities
  - MS00-083: Netmon Protocol Parsing Vulnerability
  - CVE-2004-0054: Multiple vulnerabilities in the H.323 protocol implementation for Cisco IOS 11.3T through 12.2T
- Path **Parsing** Vulnerabilities
  - MS00-017: DOS Device in Path Name Vulnerability
  - MS00-078: Web Server Folder Traversal Vulnerability

All these examples imply that any parser can have such security problems.

Recommendations

Based on learnings from the historical bug reports

- **Combing Operation** (to crack down on known security bugs)
  - Treat every security bug report as important regardless of whether it affects your or another company software and dissect the bug nature to take appropriate mitigation actions.
  - Thoroughly go through the historical bug records in the CVE databases (cvedetails.com and cve.mitre.org) or similar vendor databases, including the exploit databases (exploit-db.com), to identify all kinds of known bugs in your applications.
- **Attack Vector Database** (Create and keep it up-to-date)
  - Keep the database updated with the intel obtained through previous step regardless of whether the bug affects your or another company's software.
  - Refer to the database for identifying potential risks in your existing application and during future design changes.
- **Identify All Bug Variants** (across all applications you support)
  - Upon identifying a bug in a particular application, identify all instances and variants across the same application and any other applications you support to apply appropriate mitigation consistently.

No.2 - Learnings from the past

Learnings from the way memory corruption bugs have been brought under control in OS, Web Browsers and OS-Native Apps

Typical Exploit and Defense In Depth

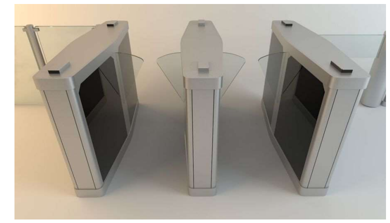
(Windows Edition)

Typical Exploit Building Blocks

- Trigger the bug
- Indirect Jump or call
- NOPSLED
- ROP Chain
- Shellcode (Payload)

Targeted Mitigation (behavioural v/s non-behavioural checks)

- Trigger the bug
- ASLR → Address Space Randomisation
- Indirect Jump or call (jmp / call / Stack Pivot)
- CFG / XFG → Prevent indirect jump or call. *Note: CFG is deprecated, and an improved version called XFG will be introduced in future releases of Windows.*
- NOPSLED
- ACG → Prevents the ability to allocate new executable memory
- ROP Chain
- Shellcode (Payload)
- DEP / NX → Mark memory pages as non-executable



Non-Behavioural Check

Behavioural Check

Behavioural v/s Non-Behavioural Mitigation

Targeted Exploit Mitigation

(Windows Edition)

Windows 10 Mitigation	Available under exploit protection
Arbitrary code guard (ACG)	yes
Block remote images	yes
Block untrusted fonts	yes
Data Execution Prevention (DEP)	yes
Export address filtering (EAF)	yes
Force randomization for images (Mandatory ASLR)	yes
NullPage Security Mitigation	yes (Included natively in Windows 10)
Randomize memory allocations (Bottom-Up ASLR)	yes
Simulate execution (SimExec)	yes
Validate API invocation (CallerCheck)	yes
Validate exception chains (SEHOP)	yes
Validate stack integrity (StackPivot)	yes
Certificate trust (configurable certificate pinning)	Windows 10 provides enterprise certificate pinning
Heap spray allocation	Ineffective against newer browser-based exploits; newer mitigations provide better protection. See Mitigate threats by using Windows 10 security features for more information
Block low integrity images	yes
Code integrity guard	yes
Disable extension points	yes
Disable Win32k system calls	yes
Do not allow child processes	yes
Import address filtering (IAF)	yes
Validate handle usage	yes
Validate heap integrity	yes
Validate image dependency integrity	yes

Windows 10 mitigation for various known exploit techniques

https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/exploit-protection?view=o365-worldwide

- Modern Operating Systems and Web Browsers focuses on killing all known techniques used in an exploit
- The list includes both behavioural and non-behavioural checks

Web-based Application Mitigation

- In Web-based applications, the widely used mitigation techniques primarily focus on non-behavioural checks against attacks.

  Example: Input Validation, Output Escaping, Parameterized Queries etc

- There is limited or no focus on introducing behavioural based mitigation

Behavioural based checks

Introducing Behavioral Based Checks

(in applications and software)

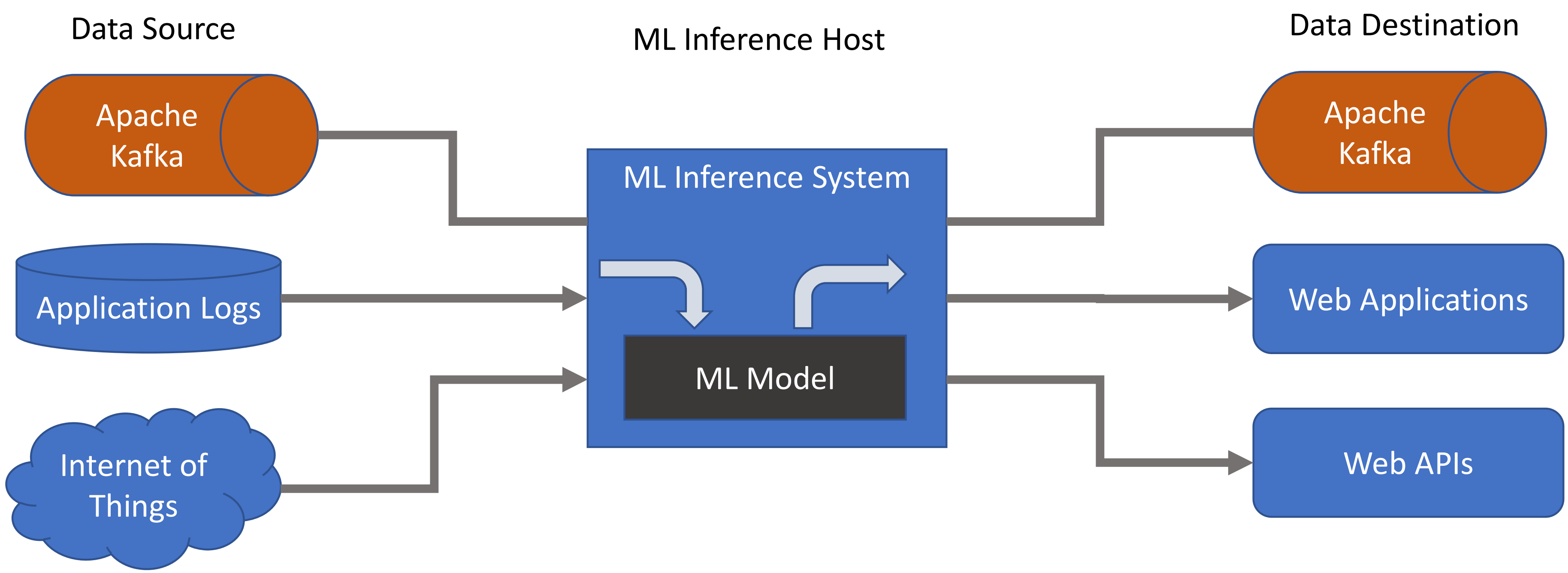
Image Source: chess.com

- An adversary can only make a finite set of moves
- Technically applications or software can be programmed to analyse infinite moves of an adversary and respond accordingly
- Integrating Machine Learning (ML) with your critical application infrastructure can do such tasks with much ease.
- ML/AI Technology has matured significantly over the years.
- Any seasoned developer can leverage ML/AI technology to integrate with applications.

Machine Learning for applications and software

Integrating Machine Learning

(in applications and software)



A simple design of ML integration with application

Recommendations

Based on learnings from the OS and Browser mitigation

- **Introduce Machine Learning (ML)**
  - Aside from the standard mitigation, introduce ML/AI technology to build behavioral checks within your application
  - Train the ML to monitor behaviours and any deviations in use cases
- **Tackling 0-days!!! Is it practical? Yes – To a larger extent**
  - Refer to CVEs, exploit databases and other product vendors security advisory, to track the nature of bugs.
  - Map those bugs with your products/applications and address them if there are similar nature bugs
  - Train the ML/AI to analyse and understand the nature of legit IN and OUT traffic. Any deviation must be blocked and inspected.
  - While achieving 100% resilience against 0-days may not seem practical. Still, with comprehensive defense-in-depth and leveraging ML, 0-days exploitation can be made very difficult to the extent that it becomes nearly impossible.

The Misconception

The Silver Bullet In Software Security Engineering

The Software Security Engineering Lifecycle Timeline

The Paradigm Shift

(in Software “Security” Engineering)

Timeline: 1985 | 1988 | 1999 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2009 | 2011 | 2012 | 2013 | 2014 | 2015 | 2017 | 2019 | 2021

Waterfall

Initial Industry Adoption

Security in Waterfall (Secure SDLC)

1988: NIST SP 500-153 - Guide to Auditing for Controls and Security

2002: GIAC Paper - Security in the SDLC by Larry G

2004: IEEE Publication: Software Security by G. McGraw

2004: The OWASP Testing Project v1.0

2006 (May): Microsoft Secure Development Lifecycle by Michael Howard and Steve Lipner

2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2009 | 2011 | 2012 | 2013 | 2014 | 2015 | 2017 | 2019 | 2021

Agile

Introduced

Security in Agile (Secure SDLC)

2005 (Dec): Secure Software Development Life Cycle Processes by Noopur Davis (Brief mention of Security in Agile)

2006 (May): Microsoft Secure Development Lifecycle by Michael Howard and Steve Lipner

2006 (Aug): Department of Homeland Security - Security in the Software Lifecycle

2007 | 2009 | 2011 | 2012 | 2013 | 2014 | 2015 | 2017 | 2019 | 2021

DevOps

Ideation | Introduced | Initial Industry Adoption

Security in DevOps (DevSecOps)

2012 (Jan): DevOpsSec: Creating the Agile Triangle (Gartner)

2012 (Apr): DevOpsSec Applying DevOps Principles to Security (DevOpsDays Presentation)

2014 (Mar): OWASP Presentation - Continuous Security Testing in a DevOps World

2015 (Oct): AWS re:Invent - Architecting for End-to-End Security in the Enterprise

The Misconception: Security is better with DevOps

The Paradigm Shift and

The Rise In Misconception

- Over the last few years, there has been a significant rise in the popularity of DevSecOps.
- However, without proper clarity on when to go for DevSecOps, there has also been an increasing misconception about it.

Snippets of Statements Extracted From Various Online Sources:

- DevOps is better with security and security is better with DevOps
- With DevOps, security gets to be introduced early in the development cycle and this minimizes risks massively
- Apps Built Better: Why DevSecOps is Your Security Team's Silver Bullet

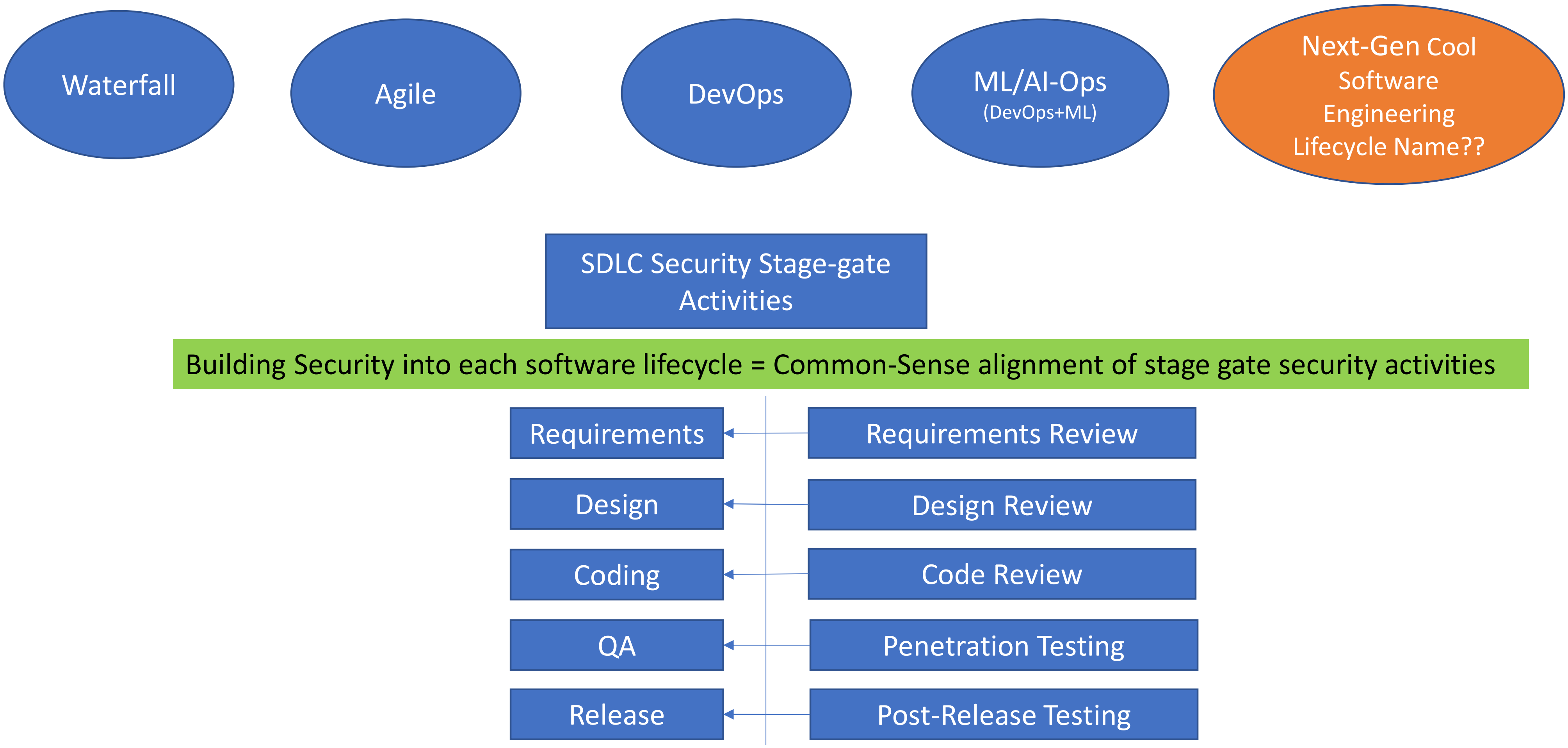
So, What Is Wrong With Such Statements?

- These statements promote in a way that Secure SDLC works best only with DevOps
- Similar statements can be found in several articles scattered all over the internet
- While promoting DevSecOps is essential, overhype can be misleading

The common-sense security activities alignment with software engineering stage gates

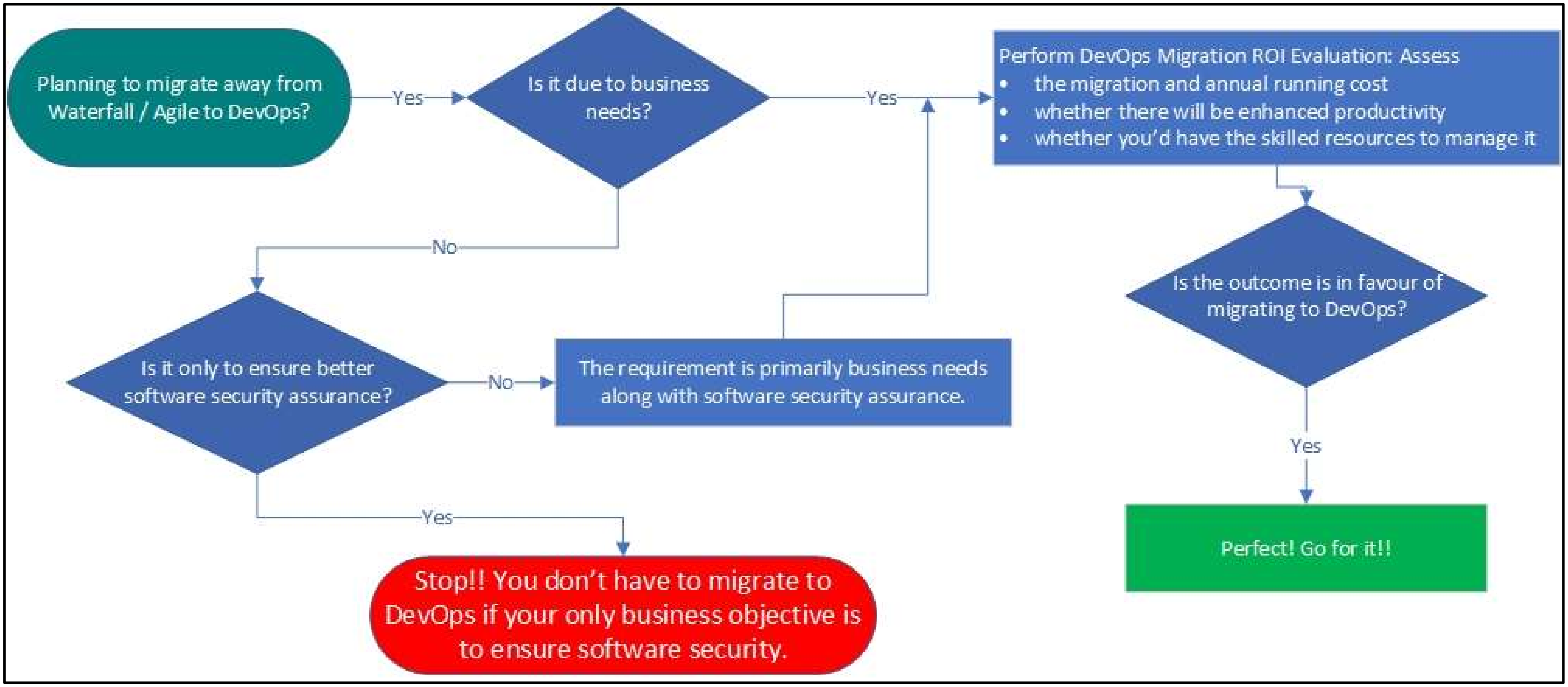
Applying Common-Sense Security

In Each Engineering Lifecycle



Decision making flow chart to determine whether to go for DevSecOps

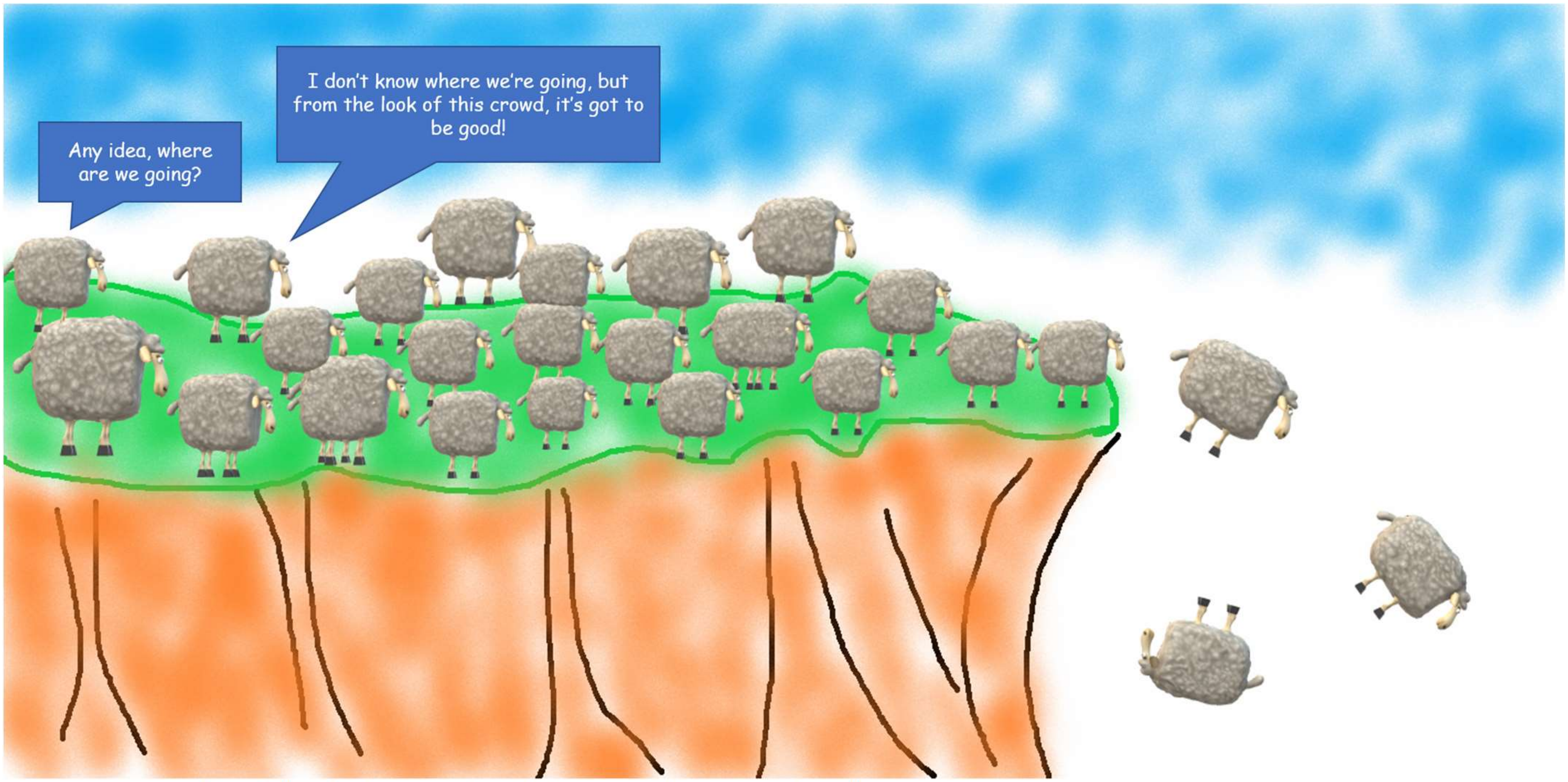
Migrating to DevOps / DevSecOps?



However, if you are migrating to DevOps because you thought or heard that the entire industry is migrating toward it, then it is not a rational decision.

The Herd Mentality

(Going with the flow without rational thinking)



Handcrafted by Debasis Mohanty using MS Paint 3D (Graphic inspired by an existing photo available somewhere online)

Building Security into the SDL

is always explicit, not implicit

- Building security into the software engineering lifecycle (Waterfall, Agile or DevOps) is always explicit, not implicit.

- There is no such Silver Bullet in Software Security Engineering

- The level of software security assurance largely depends on
  - how thorough the security assessment is done at each stage gate and
  - whether the vulnerabilities are mitigated timely

- A fixed set of common-sense security activities exists that remains the same across all types of development methodologies.

Final Words

- Treat all known security vulnerabilities as a pandemic, especially if they have been around for over decades.
- No one wants Covid-19 to last for the next 20 years. The same feelings apply to known security bugs.
- If some organisations here take away the suggestions to eradicate known bugs in your applications and achieve success in eliminating them, then spread the word and talk about your success.
- Your organisation's success story on eliminating all known bugs will inspire other organisations and potentially lead to a global ripple effect.
- Let's reassess the state of known security bugs in about 20 years from now!!! ☺

Thanks for listening to this talk!!

Q&A

Questions