2020年网络安全状况透视

翻译

weixin_26636643 于 2020-08-13 16:50:04 发布 525 收藏
文章标签： java python
原文链接：https://blog.usejournal.com/state-of-cybersecurity-2020-perspective-9998d8c69667

In information security, organizations may both feel secure when they are not, and insecure when they are actually secure given that this function is both normative and descriptive. As Schneier states, *security risk is both a subjective feeling and an objective reality, and sometimes those two views are different so that we fail acting correctly. Assuming that people act on perceived rather than actual risks, we will sometimes do things we should avoid, and sometimes fail to act like we should.*

在信息安全中，鉴于此功能既具有规范性又具有描述性，组织可能会在不安全时感到安全，而在实际上是安全时感到不安全。 正如Schneier所说， *安全风险既是主观感觉又是客观现实，有时这两种观点是不同的，因此我们无法正确采取行动。 假设人们是根据感知的风险而不是实际的风险行事，那么我们有时会做一些我们应该避免的事情，有时甚至会表现得不尽如人意。*

What is commonly known as security metrics still seems to be in a state of ideas about best-practice rather than scientific examination. Hence security metrics would require to quantify risks in systems involving a model that takes *decision maker, infrastructure assets, environment and aggregated data* into consideration.

所谓的安全度量标准似乎仍处于关于最佳实践而不是科学检查的状态。 因此，安全度量将需要量化涉及模型的系统中的风险，该模型将*决策者，基础架构资产，环境和汇总数据*考虑在内。

Let’s indulge in this speculative exercise to predict how 2020 will shape itself to either converge toward or possibly diverge from this model described above.

让我们沉迷于这种投机活动中，以预测2020年将如何塑造自己，以趋向于或可能偏离上述模型。

＃1：(行为经济学)将观点从暂时性问题转变为持续性威胁 (#1: (Behavior Economics) Shifting perspective from temporary problem to persistent threat)

Security assessments are typically conducted in reaction to a reported breach or failure to meet compliance. An executive, in such circumstances may attribute a good security posture or passing a compliance check to efficient organizational practices or great decision making in hiring. However, correlation does not always equal causation.

安全评估通常是针对所报告的违反或未达到合规性而进行的。 在这种情况下，高管可能会将良好的安全状态或通过合规性检查归因于有效的组织实践或雇用方面的重大决策。 但是，相关并不总是等于因果关系。

As per a cost study done by the Ponemon Institute, the average time it takes to detect a data breach was 206 days. The feedback delay in decision making is imperative. Unfortunately, stealthy hackers will not try to hack a company in a prescriptive fashion, and the struggle with understanding feedback delay can be seen through cognitive biases and faulty heuristics.

根据Ponemon Institute进行的一项成本研究，发现数据泄露所需的平均时间为206天 。 决策中的反馈延迟势在必行。 不幸的是，隐秘的黑客不会尝试以规定的方式对一家公司进行黑客攻击，并且可以通过认知偏见和错误的试探法来理解理解反馈延迟的努力。

Another bias exasperating this belief is the sample bias. The sample bias essentially is one seen in statistical analysis: a poor sample size will skew the results to a non accurate conclusion. If a company executive bases security spending decisions based on a sample bias made up of current data or data derived from cohort companies, then the decision may not be made based on accurate data.

另一个使这种信念恶化的偏见是样本偏见。 样本偏差本质上是在统计分析中看到的一种：样本量不足会导致结果不准确。 如果公司高管根据由当前数据或同类公司衍生的数据构成的样本偏差来制定安全支出决策，则可能无法基于准确数据做出决策。

It remains a pessimistic truth that vulnerabilities are inevitable, but it is one that executives must understand and implement retroactively in order to manage risk and lower the chances of expensive breaches occurring.

漏洞是不可避免的，这仍然是一个悲观的事实，但高管必须回溯地理解和实施该漏洞，以管理风险并降低发生昂贵的违规事件的机会。

Read `Decision-Making and Biases in Cybersecurity Capability Development` if this topic piques your attention.

如果该主题引起您的注意，请阅读` 网络安全能力开发中的决策和偏见 '。

#2：(行为经济学)自我服务的买家出现在柠檬市场中 (#2 : (Behavior Economics) The self-serving buyer emerges in a market of lemons)

`The Market for Lemons` is a seminal article written by George Akerlof in 1970, which aims to explain some of the market failures derived from imperfect information. Akerlof gave a new explanation for a well-known phenomenon: the fact that cars barely a few months old sell for well below their new-car price. The market itself is composed of two types of cars: *those that are being sold in good faith and those that are being sold off because they are known to be unreliable('lemons')*.

“ 柠檬市场 ”是乔治·阿克洛夫(George Akerlof)在1970年撰写的开创性文章，旨在解释由于信息不完善而导致的一些市场失灵。 阿克洛夫(Akerlof)对一个众所周知的现象给出了新的解释：一个事实，即仅仅使用了几个月的汽车的售价远低于其新车的价格。 市场本身由两种类型的汽车组成： *一种是善意出售的，另一种是因为众所周知的不可靠而被出售的(“ lemons”)* 。

Akerlof's model was simple but powerful. If buyers could tell which cars are lemons and which are not, there would be two separate markets: a market for lemons and a market for high-quality cars. **But there is often asymmetric information: buyers cannot tell which cars are lemons, but, of course, sellers know.** Therefore, a buyer knows that there is some probability that the car she buys will be a lemon and is willing to pay less than she would pay if she were certain that she was buying a high-quality car. This lower price for all used cars discourages sellers of high-quality cars. Although some would be willing to sell their own cars at the price that buyers of high-quality used cars would be willing to pay, they are not willing to sell at the lower price that reflects the risk that the buyer may end up with a lemon. Thus, exchanges that could benefit both buyer and seller fail to take place and efficiency is lost. The key insight is that buyers are unwilling to pay a premium for quality they cannot measure, which leads to market flooded with low quality products.

阿克洛夫的模型虽然简单但功能强大。 如果买主能够分辨出哪些车是柠檬，哪些不是，那将有两个不同的市场：柠檬市场和高品质汽车市场。 **但是，通常会有不对称的信息：买家无法分辨哪些车是柠檬，但卖家当然知道。** 因此，购买者知道，她购买的汽车很有可能是柠檬，并且愿意支付比确定自己购买的高质量汽车要少的价格。 所有二手车的这种较低价格阻碍了高品质汽车的销售商。 尽管有些人愿意以购买高质量二手车的买家愿意支付的价格出售自己的汽车，但他们不愿意以较低的价格出售，这反映出买家最终可能会得到柠檬的风险。 因此，无法使买方和卖方均受益的交易会发生，效率就会丧失。 关键的见解是，购买者不愿为无法衡量的质量付出溢价，这导致市场充斥着劣质产品。

This information asymmetry is predominant in the cybersecurity landscape. The overwhelming number of vendors with pointed solutions, FUD injected snake-oil tactics makes it far more difficult for CISOs and executive leadership teams to make an informed decision on which solutions are right for their organizational needs. This is evidenced by a survey conducted by HelpNet security and this article published by Wall Street Journal.

这种信息不对称在网络安全领域占主导地位。 大量具有针对性解决方案的供应商，FUD注入的“蛇油”策略使CISO和执行领导团队很难做出明智的决定，以决定哪种解决方案适合其组织需求。 HelpNet安全机构进行的一项调查以及《华尔街日报》发表的这篇文章证明了这一点。

In reaction, vendors begin to optimize their soft '**signals**' in order to stand apart by tweaking strategies in order achieve recognition and placement in a popular analyst's magic quadrant. Based on a quadrant placement, prospects react by buying up too many tools that introduce complexity, overhead, and lots of management time, with little impact to their actual security posture.

作为回应，供应商开始优化其软“ **信号** ”，以便通过调整策略脱颖而出，从而在知名分析师的魔力象限中获得认可和地位。 根据象限的位置，潜在客户会购买过多的工具，这些工具会带来复杂性，开销和大量管理时间，而对实际安全状况的影响很小。

The existence of information asymmetry does not necessarily impede the emergence of an **informed and influential buyer**. In reality, we as buyers have to unfortunately rely on a variety of mediocre signals to differentiate good security products from bad. In the upcoming decade we will perhaps witness a seismic shift in definition of **strong** signals to validate to vet and control vendor sprawl and help inform the decisioning process.

信息不对称的存在并不一定会阻碍一个**有见识和影响力的买方的出现** 。 实际上，不幸的是，作为买家的我们不得不依靠各种平庸的信号来区分安全产品与不良产品。 在即将到来的十年中，我们可能会目睹**强**信号的定义发生**巨大**变化，以验证审核和控制供应商的蔓延并帮助告知决策过程。

Read Phishing for Phools: The Economics of Manipulation and Deception if this topic piques your attention.

如果该主题引起您的注意，请阅读网络钓鱼诱骗用户：操纵和欺骗的经济学 。

＃3：(行为经济学)利用前景理论进行网络安全 (#3 : (Behavior Economics) Prospect theory harnessed for cybersecurity)

Choosing between two alternatives often involves risk, such as whether you should spend your money on a new car or a used car. Each choice is like two sides of a coin: there is risk of losing something (a loss) and an opportunity of getting something (a gain).

在两种选择之间进行选择通常会带来风险，例如您应该将钱花在新车还是二手车上。 每个选择都像硬币的两个方面：存在失去某物的风险(损失)和获得某物的机会(收益)。

These types of choices are determined by **a way of thinking**, which influence the how we evaluate the expected results of our decisions, and consequently, the choices we make.

这些类型的选择是通过一**种 思维 方式**确定的，这会影响我们评估决策预期结果的**方式** ，进而影响我们做出的选择。

> **Loss Aversion** — For us humans, losing something hurts more than gaining it. This happens because we are loss averse.
>
> **损失厌恶** -对我们人类来说，失去某些东西比获得它所遭受的伤害更大。 发生这种情况是因为我们厌恶损失。

Diminishing Sensitivity —People become less sensitive to changes as the value of the amounts rise. If your salary of $5000 is cut by $1000, you will feel it more than you would if your salary was $8000, but less than you would if you had a salary of $2000, though the difference ($1000) is the same in all three cases.

灵敏度降低-随着金额的增加，人们对变化的**敏感性**降低。 如果将您的$ 5000的薪水减少$ 1000，您会感觉比您的薪水是$ 8000的感觉更多，但会比拥有$ 2000的薪水感觉要少，尽管这三种情况的差额($ 1000)相同。

Reference Point — Your decisions about things are highly influenced by where you started, i.e. your Reference Point. For example, if you want to judge how bright the lights are in the room, then the amount of light present from where you just came from will affect how bright you perceive the lights to be where you are now, e.g., darkened movie theater vs. a sun room.

参考点 -您对事物的决定在很大程度上取决于您的起点(即参考点)。 例如，如果您要判断房间里的灯光有多亮，那么来自刚来自的地方的光量将影响您感知到的灯光的亮度，例如黑暗的电影院与阳光房

How do these three principles of economics defined by Nobel-prize winning professor Daniel Kahneman and Amos Tversky relate to actors in the cybersecurity realm?

诺贝尔奖获得者Daniel Kahneman和Amos Tversky所定义的经济学的这三个原理与网络安全领域的参与者有何关系？

The cloud computing paradigm and open source movement has had a huge impact on the software supply chain. In this model, resources, such as network, compute, and storage, are consumed as utilities, accessible via public or private network connections. In addition, applications in itself are composed of a transitive supply chain (your code co-mingled with open source artifacts, frameworks, SaaS vendor SDKs, etc).

云计算范例和开源运动对软件供应链产生了巨大影响。 在此模型中，网络，计算和存储等资源作为实用程序使用，可以通过公共或专用网络连接进行访问。 另外，应用程序本身由传递性供应链组成(您的代码与开放源代码工件，框架，SaaS供应商SDK等混合在一起)。

Due to this distributed nature of ownership, **loss aversion, diminishing sensitivity and reference point** differ across this supply chain.

由于所有权的这种分散性，在整个供应链中， **损失规避，敏感性降低和参考点都**不同。

> What if a system or platform emerges that can aggregate these metrics across the entire supply chain to quantify risk?
>
> 如果出现一个可以在整个供应链中汇总这些指标以量化风险的系统或平台，该怎么办？

Look no further than, Kelly Shortridge's excellent writeup on Behavioral Models of InfoSec: Prospect Theory.

凯利·肖特里奇 (Kelly Shortridge)关于“ InfoSec行为模型：前景理论 ”的出色著述不胜枚举。

Lastly researchers from Purdue University have laid the groundwork for a method to improve cybersecurity for large-scale systems like the power grid and autonomous military defense networks by harnessing prospect theory (as illustrated below)

最后，普渡大学的研究人员为利用前景理论来改善大型系统(如电网和自主军事防御网络)的网络安全方法奠定了基础(如下所示)

Read Prospect Theory: For Risk and Ambiguity if this topic piques your attention.

如果本主题引起您的注意，请阅读《 前景理论：风险与歧义》 。

＃4 ：(开发人员趋势)Rust语言将被广泛使用 (#4: (Developer trends) Rust language will be widely used)

There is a lot of interest these days in securing computer systems. This interest follows from the highly publicized data breaches, denial of service attacks, RCEs and system hijacks. In response, security companies are proliferating, selling defense in depth solutions. There is also a regular call for more “cybersecurity professionals” to help sustain these solutions.

这些天来，人们对保护计算机系统非常感兴趣。 这种兴趣来自高度公开的数据泄露，拒绝服务攻击，RCE和系统劫持。 作为响应，安全公司正在激增，销售深度防御解决方案。 经常需要更多“网络安全专业人员”来帮助维持这些解决方案。

Programming courses typically focus on how to use particular languages to solve problems efficiently. Functionality is obviously paramount, with performance an important secondary concern. Performance is measured against SLAs and engineers are incentivized to meet pre-set criteria. But in today’s climate shouldn’t security be at the same level of importance as performance?

编程课程通常着重于如何使用特定语言有效解决问题。 功能显然是最重要的，而性能是次要的重要考虑因素。 根据SLA评估绩效，并激励工程师达到预设标准。 但是，在当今的气候下，安全性不应该与性能同样重要吗？

Software artisans need to understand how a bug in a program can be turned into a security vulnerability, and how to stop it from happening. It is imperative for them to identify the conditions that make that vulnerability possible, and developing a defense that eliminates those conditions. For the latter we focus on language properties (e.g., type safety) and programming patterns (e.g., input validation, enforcement, etc).

软件技术人员需要了解如何将程序中的错误转变为安全漏洞，以及如何阻止该漏洞的发生。 他们必须确定使该漏洞成为可能的条件，并开发消除这些条件的防御措施。 对于后者，我们专注于语言属性(例如，类型安全)和编程模式(例如，输入验证，强制执行等)。

For several decades system developers have been choosing C as the one and only language for programming low-level systems. Despite many advances in programming languages, databases, operating systems, hypervisors, key-value stores, web servers, network controllers and storage frameworks are still developed in C, a programming language that is in many ways closer to assembly than to a modern high-level language.

几十年来，系统开发人员一直在选择C作为唯一的低级系统编程语言。 尽管编程语言取得了许多进步，但是数据库，操作系统，虚拟机管理程序，键值存储，Web服务器，网络控制器和存储框架仍是用C语言开发的，这种编程语言在许多方面更接近于汇编，而不是现代的高级语言。语言。

Rust is a system programming language that offers a practical and safe alternative to C. It is unique in that it enforces safety without runtime overhead, most importantly, without the overhead of garbage collection. With modern compiler technology, Rust is able to offer safety *and* performance in a somewhat ergonomic package.

Rust是一种系统编程语言，为C提供了一种实用且安全的替代方法。它的独特之处在于它可以在没有运行时开销(最重要的是，没有垃圾收集开销)的情况下增强安全性。 借助现代的编译器技术，Rust可以在符合人体工程学的包装中提供安全性*和*性能。

Looking ahead, Rust will become the de-facto choice of systems programming.

展望未来，Rust将成为事实上的系统编程选择。

＃5 ：(开发人员趋势)Web Assembly(WASM)将被广泛使用和滥用 (#5: (Developer trends) Web Assembly (WASM) will be widely used and abused)

WASM is designed to allow browsers to run code in a sandbox, isolating the impact of vulnerabilities in WASM code from the rest of the browser. It is a binary format currently developed and supported by all major web-browsers including Firefox, Chrome, Webkit/Safari and Microsoft Edge. This new format have been designed to be “Efficient and fast“, “Debuggable“ and “Safe” that why it is often called as the “**game changer for the web**”. WASM is Turing complete and supports many of the features you’d expect from a low-level language.

WASM旨在允许浏览器在沙箱中运行代码，从而将WASM代码中的漏洞的影响与其他浏览器隔离开。 它是当前由所有主要的网络浏览器开发并支持的二进制格式，包括Firefox，Chrome，Webkit / Safari和Microsoft Edge。这种新格式被设计为“高效，快速”，“可调试”和“安全”，因此通常被称为“ **网络改变者** ”。 WASM已经完成了Turing的功能，并支持您希望从低级语言获得的许多功能。

There is a steady and rampant adoption of WebAssembly in various domains:

WebAssembly在各个领域都得到了稳定而广泛的采用：

- Web-browsers (Desktop & Mobile)
  网络浏览器(桌面和移动)
- Servers/Website (Nodejs, React, Qt, Electron, Cloudflare workers)
  服务器/网站(Nodejs，React，Qt，Electron，Cloudflare工作者)
- Video games (Unity, UE4)
  电子游戏(Unity，UE4)
- Blockchain platforms (EOS, Ethereum, Dfinity)
  区块链平台(EOS，以太坊，Dfinity)
- Linux Kernel (Cervus, Nebulet) , etc and …
  Linux内核(Cervus，Nebulet)等…

  WEAPONIZATION : Cryptojacking (Coinhive, Cryptoloot)

  自动化 ：加密劫持(Coinhive，Cryptoloot)

But keeping the browser safe from WASM code is not the same as keeping WASM code safe from itself — isolation doesn’t prevent attackers from exploiting memory-safety bugs to compromise the WASM code and any data it handles.

但是，保护浏览器免受WASM代码的侵害与保持WASM代码自身的安全性不同–隔离并不能阻止攻击者利用内存安全性漏洞来破坏WASM代码及其所处理的任何数据。

Although WebAssembly will gain steady adoption, it will also make it more likely that vulnerabilities related to memory management will occur in both WebAssembly engines and applications written in WebAssembly.

尽管WebAssembly将获得稳定的采用，但也将更可能在WebAssembly引擎和以WebAssembly编写的应用程序中发生与内存管理相关的漏洞。

Unfortunately, we can’t simply modify WASM to enforce strong memory safety by default, like past bytecodes for high-level languages.

不幸的是，默认情况下，我们不能简单地修改WASM以增强存储安全性，例如过去用于高级语言的字节码。

Patrick Ventuzelo presented a detailed study on detecting and analyzing WASM Cryptominer attack here.

帕特里克Ventuzelo介绍了检测和分析WASM Cryptominer攻击了详细的研究在这里 。

To start with

从开始

- Incorporate runtime inspection to detection WASM files as a separate file, array of bytes insider a block in your JavaScript bundle
  将运行时检查合并为一个单独的文件，将WASM文件检测为JavaScript包中的块内部的字节数组

  Detect WASM binary in runtime (*starts with the magic bytes \x00\x61\x73\x6d*)

  在运行时检测WASM二进制文件(*以 魔术字节 \ x00 \ x61 \ x73 \ x6d开头*)

  Detect WASM injection points (statically via control flow analysis or via dynamically via code injection)

  检测WASM注入点(通过控制流分析静态地或通过代码注入动态地)

  `new WebAssembly.Instance(…)`, `WebAssembly.instantiate(…)`, `WebAssembly.instantiateStreaming(…)`

  `new WebAssembly.Instance(…)`，`WebAssembly.instantiate(…)`，`WebAssembly.instantiateStreaming(…)`

#6：(产品趋势)下一代代码分析和模糊测试将成为主流，并且代码分析将指导模糊测试 (#6: (Product Trends) Next generation code analysis and fuzzing will become mainstream and code analysis will guide Fuzzing)

Software has grown in both complexity and dynamism over the years. . Evidently, the scale of present-day software development puts an enormous pressure on proactive security testing. Evaluating the security of large applications that are under active development is a daunting task

这些年来，软件的复杂性和动态性都在增长。 。 显然，当今软件开发的规模给主动安全测试带来了巨大压力。 评估正在开发中的大型应用程序的安全性是一项艰巨的任务

Static analysis is the analysis of software at compile time without executing it. Static analyzers allow various actors of the Software Development Lifecycle to proactively detect software issues such as security vulnerabilities (eg, RCE, XXE, XXS, SSRF), data leaks (CCPA, GDPR), business logic flaws and inside threat patterns. However, legacy tools provide superficial feedback and their use is often seen as "*too much pain, too little gain*".

静态分析是在编译时不执行软件的情况下进行的分析。 静态分析器允许软件开发生命周期的各个参与者主动检测软件问题，例如安全漏洞(例如RCE，XXE，XXS，SSRF)，数据泄漏(CCPA，GDPR)，业务逻辑缺陷和内部威胁模式。 但是，旧版工具提供了肤浅的反馈，它们的使用通常被视为" *痛苦太多，收益太少* "。

The next generation of code analysis will gain rampant adoption in 2020 and will be integration by default in your software assembly line. As a consequence of its efficiency and precision, the findings produced by code analysis will be leveraged by downstream engines techniques like fuzzing, WAF, etc.

下一代代码分析将在2020年得到广泛采用，并将默认情况下集成到您的软件组装线中。 由于其效率和准确性，代码分析产生的结果将被诸如模糊测试，WAF等下游引擎技术所利用。

At a high level, fuzzing refers to a process of repeatedly running a program with generated inputs that may be syntactically or semantically malformed. Among the many software vulnerability discovery techniques available today, fuzzing is gaining momentum due to its conceptual simplicity, its low barrier to deployment, and its vast amount of empirical evidence in discovering real-world software vulnerabilities. Yet, contemporary fuzzers fall short of thoroughly testing applications with a high degree of control-flow diversity.

从较高的层次上讲，模糊是指重复运行程序的过程，生成的输入可能在语法上或语义上格式错误。 在当今可用的许多软件漏洞发现技术中，由于其概念上的简单性，部署的低障碍以及在发现现实世界软件漏洞中的大量经验证据，模糊测试正在得到发展。 但是，当代的模糊测试无法充分测试具有高度控制流多样性的应用程序。

Thus, next generation code analysis engines can guide fuzzing by augmenting existing program models maintained by the fuzzer. Based on the insight that code patterns reflect the data format of inputs processed by a program, it automatically construct an input dictionary by statically analyzing program control and data flow (using Code Property Graph) . The input dictionary generated by code analyzer is supplied to an off-the-shelf fuzzer to influence input generation.

因此， 下一代代码分析引擎可以通过增加由模糊器维护的现有程序模型来指导模糊。 基于代码模式反映程序处理的输入的数据格式的见解，它通过静态分析程序控制和数据流(使用“代码属性图”)自动构建输入字典。 由代码分析器生成的输入字典将提供给现成的模糊器以影响输入生成。

Fueled by these activities, defenders have started to use fuzzing in a an attempt to discover vulnerabilities before attackers do. For example, prominent companies such as Google, Facebook and Microsoft all employ converged code analysis + fuzzing as part of their secure development practices.

在这些活动的推动下，防御者开始使用模糊测试，试图在攻击者发现之前发现漏洞。 例如，谷歌，Facebook和微软等著名公司都将融合代码分析+模糊化作为其安全开发实践的一部分。

＃7：(产品趋势)应用程序安全性将根据规范而非手工验证例程进行衡量 (#7: (Product Trends) Application Security will be measured against specification(s), not artisanal validation routines)

Systems are often developed without security in mind. This omission is primarily because developers are focusing more on trying to learn the domain, optimize time for feature delivery rather than worrying about how to protect the system. In these cases, security is usually the last thing he or she needs or wants to worry about. When the time arrives to deploy these systems, it quickly becomes apparent that adding security is much harder than just adding an authentication, 2FA layering and zoning.

系统开发时往往没有考虑安全性。 遗漏的主要原因是，开发人员将更多精力放在尝试学习领域，优化功能交付时间上，而不用担心如何保护系统。 在这些情况下，安全通常是他或她需要担心或想担心的最后一件事。 当部署这些系统的时间到来时，很快就会发现，增加安全性远不只是增加身份验证，2FA分层和分区。

演示地址

Nowadays, security is often implemented in a programmatic way, meaning a developer reacts to feedback provided during code analysis and thereafter constructs a custom validation routine to solve for a targeted problem. This of course has negative impact on reusability and flexibility of the resulting code. Especially when thinking about component oriented software development, security has to be realized outside the component's code in order to enable the usage in other software applications. The decoupling of security from the component is an important need since the security requirements vary between applications.

如今，安全性通常以编程方式实现，这意味着开发人员会对代码分析期间提供的反馈做出React，然后构造一个自定义验证例程来解决目标问题。 当然，这会对结果代码的可重用性和灵活性产生负面影响。 特别是在考虑面向组件的软件开发时，必须在组件代码之外实现安全性，以便能够在其他软件应用程序中使用。 安全性与组件的分离是一个重要的需求，因为安全性要求在应用程序之间会有所不同。

Thus, there is the need for security mechanisms that protect code and resources but which are provided and maintained outside the application in an extra layer. Such declarative security mechanisms allow greater flexibility, reusability and maintainability to the code.

因此，需要保护代码和资源但在应用程序外部的额外层中提供和维护的安全性机制。 此类声明性安全机制可为代码提供更大的灵活性，可重用性和可维护性。

Frameworks play a pivotal role in the security of an application. If every developer had to reimplement basic security features such as authorization, authentication, validation for every application, this would always include the possibility of security vulnerabilities being introduced. In that way, because frameworks can provide a lot of functionality already, they can greatly help developers, who now do not have to develop those features themselves.

框架在应用程序的安全性中起着至关重要的作用。 如果每个开发人员都必须为每个应用程序重新实现基本的安全功能，例如授权，身份验证，验证，那么这将始终包括引入安全漏洞的可能性。 这样，因为框架已经可以提供很多功能，所以它们可以极大地帮助开发人员，他们现在不必自己开发这些功能。

Majority of the popular open source frameworks (Spring, Play!, NodeJS, Django) provide declarative security via framework configuration or interfaces. However they are either under utilized or misused due to lack of awareness.

大多数流行的开源框架(Spring，Play！，NodeJS，Django)通过框架配置或接口提供声明性安全性。 然而，由于缺乏认识，它们要么未被利用，要么被滥用。

In this blog post titled Towards a concept of Security Specification for Software Supply Chain , I touched on a concept that can verify the effective utility of an open source framework via enabling/disabling declarative features and properties. This model enables OSS authors to define effective protection measures that can be leveraged by upstream supply chain.

在这篇题为“ 面向软件供应链安全规范的概念”的博客文章中，我谈到了一个可以通过启用/禁用声明性功能和特性来验证开源框架有效效用的概念。 该模型使OSS作者可以定义上游供应链可以利用的有效保护措施。

＃8：(产品趋势)动态云环境中的对抗建模 (#8: (Product Trends) Adversarial Modeling in dynamic cloud environments)

Cloud infrastructure has become both ubiquitous and more complex in the past few years. Analysts predict that by the year 2020, 60–70% of the entire IT infrastructure which includes deployed applications, technologies and services will be cloud-based. While the compute, storage capacity, networking efficiency, and hardware have received due attention and evolved with business demands, the various aspects that govern the security of a cloud infrastructure are still managed using traditional means i.e. *whether, monitoring network traffic for malicious patterns, patching known vulnerabilities (even if done regularly), and relying on the network and perimeter defense such as Firewalls, Intrusion Detection Systems, Anti-virus, tools enough to detect and thwart attacks?*

在过去的几年中，云基础架构已经变得无处不在且变得更加复杂。 分析人士预测，到2020年，整个IT基础架构(包括已部署的应用程序，技术和服务)的60-70％将基于云。 尽管计算，存储容量，网络效率和硬件已受到应有的重视并随着业务需求而发展，但仍使用传统方式来管理支配云基础架构安全性的各个方面，即*是否监视网络流量是否受到恶意模式，打补丁。已知的漏洞(即使定期执行)，并依靠网络和外围防御(如防火墙，入侵检测系统，防病毒)以及足以检测和阻止攻击的工具？*

First, the adversary, with time on their side, can spend reconnaissance effort in modeling the cloud system (situational awareness) and then carefully plan their attacks. Second, implementations of these defenses in practice are often far from ideal, thereby allowing attackers even more opportunities to exploit the system.

首先，在有时间的情况下，对手可以花费精力进行云系统建模(状况感知)，然后仔细计划其攻击。 其次，在实践中这些防御措施的实现通常远非理想之举，从而使攻击者有更多机会利用该系统。

To address the shortcomings of existing defense methods, **Moving Target Defense (MTD)** has emerged as a solution that provides a proactive defense against adaptive adversaries. The goal of MTD is to constantly shift between multiple configurations in a cyber-system (such as changing the open network ports, network configuration, rotating keys, software versions, etc.) in order to increase the uncertainty for the attacker; in effect, diminishing the advantage of reconnaissance (or situational awareness) that an attacker inherently has against traditional defense mechanisms. The shifting mechanism should be non-deterministic as otherwise, the attacker can model and adapt attack vectors accordingly. Thus, MTD techniques should always have implicit randomness built into them. Note that this randomness increases the cost for an attacker to succeed in using a zero-day attacks because it does not necessarily know which configuration of the system is in place at any particular moment.

为了解决现有防御方法的缺点， **移动目标防御(MTD)**已经成为一种解决方案，可以主动防御自适应对手。 MTD的目标是不断地在网络系统的多个配置之间切换(例如更改开放的网络端口，网络配置，旋转密钥，软件版本等)，以增加攻击者的不确定性。 实际上，这削弱了攻击者固有的针对传统防御机制的侦察(或情况感知)优势。 转移机制应该是不确定的，否则攻击者可以相应地对攻击向量进行建模和调整。 因此，MTD技术应始终具有内置的隐式随机性。 请注意，这种随机性增加了攻击者成功使用零时差攻击的成本，因为它不一定知道在任何特定时刻系统的配置正确。

MTD (as a feature) will either be subsumed as a Cloud provider function or perhaps, be a part of modern container chassis like Kubernetes and D2IQ.

MTD(作为一项功能)将被归入Cloud提供商功能，或者可能成为Kubernetes和D2IQ等现代容器机箱的一部分。

谢谢读者！ (Thank you readers!)

Of several books/videos/publications I read over the past year, here are top ones that left a lasting impression

在过去一年里我读过的几本书/录像带/出版物中，有一些给人留下深刻印象

A Codebase is an Organism — by *Kevin Simler*

代码库是一种有机体 — *凯文·西姆勒* (*Kevin Simler)*

Every Security Team is a Software Team Now — by *Dino Dai Zovi*

每个安全团队现在都是软件团队 *-Dino Dai Zovi*

The security products we deserve — by *Haroon Meer* & *Adrian Sanabria*

我们值得拥有的安全产品 *-Haroon Meer* 和 *Adrian Sanabria*

The curious case of Scale in Cyber Security — by *Vincenzo Lozzo*

网络安全规模的奇特案例 *-Vincenzo Lozzo*

The Githubification of InfoSec — by *John Lambert*

InfoSec的千篇一律 — *John Lambert*

Leverage Points: Places to Intervene in a System — by *Donella Meadows*

杠杆点：干预系统的场所 — *Donella Meadows*

Introducing the Funnel of Fidelity — by *Jared Atkinson*

介绍忠实的漏斗 - *贾里德·阿特金森* (*Jared Atkinson)*

Farnam Street editorial — by Shane Parrish

Farnam街社论 – Shane Parrish

Ness Labs editorial — by Anne-Laure

内斯实验室社论 – 安妮·劳尔 (Anne-Laure)

我期待在2020年与您一起学习新事物。新年快乐！ (I look forward to learning new things with you in 2020. Happy New Year!)

翻译自: https://blog.usejournal.com/state-of-cybersecurity-2020-perspective-9998d8c69667