**DEVELOPMENT ENVIRONMENT SECURITY REVIEW**

**VERIFACTS SERVICES PRIVATE LIMITED**

# Introduction

There is sometimes a perceived conflict between security and usability. This situation is highlighted in the case of end user devices and the environments used to support software development.

Traditional enterprise IT is often configured to prevent users from running arbitrary code (and for good reason). However, some developers will require a less constrained environment, one which allows them to install relevant tooling and test their software effectively. Such flexibility can expose a broad attack surface to would-be attackers. This can make security professionals uncomfortable.

Developers tend to be technically smart. They will find 'clever' ways to work around hindrances, if that's what it takes to get their job done. We need a situation which is both flexible and secure. This guidance outlines how you can achieve this.

# Flexible and secure

Securing your development environment is not about preventing our developers from working. It's about understanding the risks to our environments, applying technical controls where appropriate and being in a position to trust and verify legitimate usage.

The logical line between development and operations is becoming blurry. In environments where products and services are deployed through code, maintenance or operational activities can be carried out via configuration and software changes. The same individuals may often both develop and operate a service.

Where this is the case, it's important to recognize the onward impact that a compromised device may have on our services, and apply controls to help limit this exposure.

# Security gained

By performing the functions outlined in subsequent sections, we can help reduce the risk of an attacker:

* stealing sensitive information (such as encryption and access keys, passwords or knowledge of security controls)
* embedding malicious code in our project without our knowledge
* using a compromised development device as a proxy to further attack our build and deployment pipeline, through to production
* stealing code for the intellectual property it contains, or to release it publicly, causing embarrassment or degrading our security
* understanding how our sensitive applications work - a first step in the planning of an attack

# Securing a flexible system

To do their job effectively, developers often need administration or root level permissions on their working environment.

However, providing this kind of flexibility can entail a bigger impact if malware compromises their device. There are numerous ways this can happen, including spear-phishing, drive-by downloads, or accidental malware installation.

The following measures can help us to understand and reduce these risks:

* **Separate business and development functions**

Core business services such as email and document management often contain sensitive and highly prized information about our organisation and customers. If an attacker is able to compromise our development environment, they may try to pivot from this position to obtain additional benefit from their attack. Logically separating our development environment from other functions makes pivoting more difficult.

Separation can be achieved without resorting to physically different devices. For example, a user may 'browse down' from a protected environment that is more locked down, into a local virtual machine or a remote environment where they can carry out development functions in a more flexible environment.

* **Consider your development environment compromised**

If an attacker is able to compromise a developer's environment they will inherit the same level of permissions and access as that developer. Placing additional controls between our developer environments and critical systems will help to reduce this impact.

For example, the use of multi-factor authentication will make it harder for an attacker to leverage stolen credentials and access tokens. Automated security testing and a multi-person review process as part of our deployment pipeline can help catch and prevent onward impact.

* **Trust your developers, verify their actions**

People are not the weakest link, they are the first line of defence. Security-savvy individuals can sometimes be better at detecting and preventing attacks than technical controls.

Trust can be placed in individuals who are security aware and strive to do the right thing. Investing in monitoring and auditing controls will help you to verify this is happening. Examples include network monitoring to detect suspicious activity, checking patch levels and verifying software that users are installing.

# Actions

* **Provide developers with the tools and environment they need**  
  Practical tooling is essential if we want to avoid technically savvy developers finding insecure workarounds. This may sometimes mean being a bit more tolerant to the risks of a developer device being compromised and putting wider protections in place to account for this. If an IT department can respond effectivley to software requests, it may be possible to avoid giving out local administrative permissions.
* **Trust your developers, verify their actions**  
  Educate users and trust them to do the right thing. Implement functions to verify their activity, and to detect suspicious activity or potential compromise.
* **Reduce the attack surface of your developer environment**  
  Apply the [NCSC end user device guidance](https://www.ncsc.gov.uk/guidance/end-user-device-security) to our development environment. Be pragmatic in the selection of our security controls so that we don't hinder development. Consider the security mitigation technology provided by devices and operating systems when selecting our devices and environments.
* **Protect access credentials and secret keys**  
  Carefully consider how we store and handle credentials that grant access to critical services and systems. Using multi-factor authentication will mean that stolen passwords are not sufficient on their own to gain unauthorized access. Use a least privilege access model and revoke any credentials which may have been compromised, or are no longer needed.
* **Assess the impact of a compromise and apply onward controls**  
  In the event a development environment is compromised, consider the onward technical controls that will protect the products and services you produce and publish.
* **Don't operate your production systems from a development environment**  
  Managing production ('live') services from development environments can be high risk. In situations where this is required, try to limit the exposure of your production services. Consider models for [least privileged](https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/securing-privileged-access-reference-material) access management and temporary access credentials through a request process. System [management interfaces](https://www.ncsc.gov.uk/blog-post/protect-your-management-interfaces) are important to secure, but out of scope for this guidance.
* **Apply network architecture controls**  
  In the event that developer devices are compromised, constrain their onward access with wider network architecture controls that supports defense in depth. Examples may include firewall rule sets only permitting developer devices to communicate to intended services. Also consider the impact these services have on your developers. For example, non transparent proxies can hinder use of some technologies.
* **Carry out protective monitoring of your development environment**  
  We should have an idea of what legitimate use of our environment looks like. Combining this knowledge with logging from your environment can help to detect illegitimate use or potentially, a compromise. For example, are strange websites being accessed and are privileged actions being carried out during unusual hours of the day?