



---

**The nCircle Precision Metric:**  
Proactively Improving the Quality of Vulnerability Management

# Table of Contents

<b>Introduction</b> .....	<b>3</b>
<b>Specificity</b> .....	<b>4</b>
Protocols .....	4
Applications .....	4
Vulnerabilities .....	5
<b>Immutability</b> .....	<b>5</b>
<b>Precision Score</b> .....	<b>6</b>
Per Signature .....	6
Per Application or Vulnerability .....	6
Aggregate Precision Score .....	6
<b>Conclusion</b> .....	<b>7</b>

# 1. Introduction

The nCircle VERT team developed the Precision Metric in order to provide a means of quantifying the quality of IP360's vulnerability and application signatures. Our customers equate quality with accuracy—to them, the more accurate a signature, the higher its quality. However, measuring the accuracy of signatures can be difficult. Accuracy relies entirely on real-world performance. Although we can test signatures internally, no amount of lab equipment can completely replicate the real world, nor conclusively prove that a check will perform equally well outside the lab. We therefore use the Precision Metric as a framework for quantifying the likely accuracy of any given check, and to understand the logical relationship of its vulnerability and application signatures to the ideal of real-world accuracy.

There are two components to the Precision Metric: *specificity* and *immutability*.

Specificity describes how narrowly or broadly we define the condition we seek to test. The most specific check would be a test of the vulnerability via direct exploitation. A very general check might be for the existence of a broad class of application service associated with the vulnerability. Between those two extremes, signatures can evaluate secondary conditions that indicate the existence or absence of the condition we want to test. The results of this evaluation fall into different categories of specificity.

In most first-generation scanning tools, a test is usually either an actual exploit or a service/banner check. We realize from experience, however, that there are many useful shades of gray. We can often infer that a particular service is vulnerable to a given condition in an indirect, yet conclusive, manner. For example, when software vendors fix vulnerabilities, they often fix other bugs at the same time. In some cases, even with rigorous research, it may prove impossible to test for the original condition without exploiting it or being destructive to the system. However, since other software fixes were introduced at the same time, it is often possible to infer the presence of the original condition from the presence of other vulnerabilities that are meant to be patched by the same fix.

The other component of the Precision Metric is *immutability*. Immutability is a measurement of how easily a given piece of information can be altered. We gauge immutability by looking at how easily a piece of information can be modified and whether the modification will affect the proper operation of the application or system as a whole. Sometimes, changing system information can be easy: an administrator can just modify behavior by using a graphical interface. At other times, it can be difficult: an administrator might have to modify the firmware of a device to obfuscate or change a piece of system information. In many cases, it can even be problematic to change information because the system relies on it in order to function properly.

Changing banners via configuration files is quite easy to do, as opposed to modifying the source code or manually editing an application binary. An application banner is a default response to a remote connection. These often provide obvious, plain-text information about the type and version of the software. When we refer to the weakness of "banner checks", we are pointing out the inherent mutability of these responses – most modern applications now provide ways to eliminate or modify these responses through easily available configuration options. For example, the most common web server, Apache, makes it trivial to modify the server banner to be incorrect or non-existent. If we relied on this highly mutable information, it could lead to the server being misidentified (false positive) or not identified at all (false negative). On the other hand, there are many applications that are much more difficult to configure in this way, such as hardware wireless access points and some printers. If modifying the banner requires a soldering iron, it is highly unlikely that the banner will have been changed. While it is possible to modify hardware to display misleading information or no information at all, it is extremely rare. The more difficult it is to modify information, the more immutable it is.

The Precision Metric describes the precision of our signatures, but does not always guarantee accuracy in the real world. While a very precise signature will generally be more accurate than an imprecise one, the conditions in real-world, customer environments are not guaranteed to behave in a predictable way. Our measurement of precision gives us a reasonable assurance that a signature is likely to be accurate. Historically, we have had some extremely precise signatures that have caused false positives or false negatives, but we generally see more issues with our imprecise signatures.

## 2. Specificity

*Specificity* is a measurement of how closely a given test describes a particular protocol, application, or vulnerability. Specificity is measured on a scale of 1 to 5, with 5 being the most specific.

The meaning of a particular level varies depending on whether it is a protocol, an application, or a vulnerability that is being measured. The charts below will aid in determining the most suitable Specificity level for a given situation.

### 2.1. Protocols

---

#### **Level 1: Open Port**

The port on which this protocol usually exists is open.

Example: Port 23 is open.

#### **Level 2: Returns Data**

The port returns data that does not necessarily match the content or structure of the protocol.

Example: Sending data to port 23 results in returned data.

#### **Level 3: Returns Structure or Content**

The port returns data whose structure or content, but not both, resembles the protocol.

Example: Port 23 returns a three-digit number and a string of bytes terminated by a CRLF. Or, port 23 returns the number "220" and the hostname of the server.

#### **Level 4: Returns Structure and Content**

The port returns data whose structure *and* content resembles the protocol.

Example: Port 23 returns the number 220 followed by a space, followed by the hostname of the server, followed by a space, followed by ESMTP, and terminated by CRLF.

#### **Level 5: Protocol Interaction**

It is possible to repeatedly send protocol commands to the port and consistently receive appropriate data.

Example: Conducting an SMTP "dialogue" with the remote server.

### 2.2. Applications

---

#### **Level 1: Protocol Identified**

The protocol used by the application is present.

Example: HTTP exists.

#### **Level 2: Application Identified**

The general application has been identified, although no specific information about the application version is available.

Example: Apache exists.

#### **Level 3: Application Variant**

A major variant of the application has been identified.

Example: Apache 1.3 exists.

#### **Level 4: Group Unique Behavior**

The application is known to be one of a limited number of minor variants on the basis of a specific behavior common to those particular variants and no others.

Example: Apache 1.3.3[0-3] exists.

#### **Level 5: Unique Behavior**

A single, specific variant of the application has been identified on the basis of a unique behavior that is only exhibited by that particular variant of the application.

Example: Apache 1.3.33 exists.

### **2.3. Vulnerabilities**

---

#### **Level 1: Application Exists**

Some version of the application in which the vulnerability is present has been determined to exist.

#### **Level 2: Application Version Exists**

A particular version of the application in which the vulnerability is present has been determined to exist.

#### **Level 3: Independent Attribute**

The application has some attribute that is likely to exist when the vulnerability exists, although that attribute is not the vulnerability itself. Changing this attribute will have no effect on the status of the vulnerability.

#### **Level 4: Dependent Attribute**

The application has some attribute that is likely to exist when the vulnerability exists, although that attribute is not the vulnerability itself. Changing this attribute *will* have an effect on the status of the vulnerability.

#### **Level 5: Direct Exploitation**

The vulnerability is shown to exist by exploiting it directly.

## **3. Immutability**

Like Specificity, there are five levels of Immutability, with 5 being the most immutable.

The chart below will aid in determining the most suitable Immutability level for a given situation.

#### **Level 1: Dynamic Content**

The ability to change content is inherent to the system; the information can be reasonably expected to differ every time it is examined.

#### **Level 2: Configuration Option**

Changing information requires configuration files or system settings to be changed. Although this information is designed to be changed easily, it is not likely to be changed especially often.

#### **Level 3: Source Edit**

Changing information requires changing the source code and re-compiling.

## **4. Precision Score**

### **4.1 Per Signature**

---

The Precision of a given signature is simply the product of the Specificity level and the Immutability level multiplied by 4, giving a Precision Score from 0 to 100.

Precision = (Specificity x Immutability) x 4

#### **Example:**

V1545 – Microsoft IIS Unicode Directory Traversal

The vulnerability can be checked non-invasively via direct exploitation. It would therefore have a Specificity score of 5. It is impossible to change the behavior of the vulnerable system without changing the behavior of Unicode itself. This vulnerability would therefore have an Immutability score of 5. This gives a Precision of:

$(5 \times 5) \times 4 = 100$

### **4.2 Per Application or Vulnerability**

---

Vulnerabilities and applications also have their own Precision Scores. This value will be calculated by taking the average of the Precision Score values for each of the non-deleted, verified signatures within the vulnerability or application.

### **4.3 Aggregate Precision Score**

---

Each vulnerability and application has an *Aggregate Precision Score*. The Aggregate Precision Score is a weighted sum of the Precision Scores of all the application tree nodes from the application root node to the child node (which is itself either an application or a vulnerability).

Specifically, the formula used is:

SUM (Precision Score / Node Level) of each node from child to root.

Node Level refers to the level of the node within the application tree. The top level node has a Node Level of 1; its immediate child has a Node Level of 2, its grandchild has a Node Level of 3, and so forth. All protocol level nodes have a Node Level of 1. A vulnerability has a Node Level that is one point higher than the Node Level of the application node to which it is bound.

For vulnerabilities that have per-signature application bindings, the Aggregate Precision Score is calculated as the average Aggregate Precision Score for each signature in the vulnerability.

Similarly, the Aggregate Precision Score of a vulnerability that is bound to multiple applications is calculated as the average Aggregate Precision Score along *each path*. For example, if a vulnerability is bound to three applications, then three Aggregate Precision Scores are independently calculated. The three Aggregate Precision Scores are then averaged to calculate the Aggregate Precision Score of the vulnerability.

## 5. Conclusion

The goal of using the Precision Metric is to increase the quality of vulnerability and application detection signatures. The Precision Metric gives a quantitative value that guides the creation of quality signatures. The better the signatures, the better the product, and more accurate the information we can provide to our customers. Our customers need actionable intelligence, and a large part of gathering intelligence is determining just how good it is, and if necessary, making it better. With the Precision Metric, we can quantify the value of the intelligence we provide and use it proactively to improve the quality of our signatures, rather than waiting for defects to occur.

## Authors

Minoo Hamilton  
Gauthaman Ravindran

### **Additional Contributors:**

Mike Murray  
Tim Keanini  
Christian Hunt  
Joel Weisz  
Paul Miseiko

## About nCircle

nCircle is a leading provider of enterprise-class vulnerability and risk management solutions. Global enterprises and government agencies rely on nCircle's proactive security solutions to identify, measure, manage and reduce security risk on their worldwide networks. nCircle has won numerous industry awards for its rapid growth, innovation and technology leadership and has been named one of the top 100 best places to work in the San Francisco Bay Area. nCircle is headquartered in San Francisco, California, with regional offices throughout the U.S. and in London, Toronto and Tokyo. Additional information about nCircle is available at [www.ncircle.com](http://www.ncircle.com).

### **nCircle Network Security**

101 2<sup>nd</sup> St. Suite 400  
San Francisco, CA 94105  
(888) 464 2900  
[www.ncircle.com](http://www.ncircle.com)