WHITE PAPER

C++ Super Glue for Anti-malware Applications

BENNY CZARNY

OPSWAT®

SECTION **0.0**

Abstract

The growing number of anti-malware applications present interoperability and security management issues to IT professionals and programmers seeking a common language to classify and manage anti-malware features.

Integrators conduct research for anti-malware application interfaces; this research is usually time consuming and includes looking into dll header files, Command Line interface (CLI), C++ or COM API (if available) as well as other techniques, such as modifying registry keys, files and process monitoring.

This research is successful when the number of managed applications is limited to a few, but becomes an engineering challenge as the quantity of applications to be managed increases. The challenges and opportunities surrounding application management will be addressed during this session.

This session will begin by identifying integration challenges and will then introduce the concept of an object-oriented "universal language" which serves as the basis for an interoperability technology. I will then demonstrate how several different security vendors in the field of remote access, network access control and filtering technologies, successfully implemented this language. The session will continue by introducing an "object-oriented analysis" approach for anti-malware application management. I will identify attributes (properties) and operations (methods) associated with anti-malware applications and will discuss the advantages of object-oriented architecture and other techniques of solving issues related to anti-malware application management.

SECTION **1.0**

The Anti-malware Application Chaos

In its survey of 616 US IT security professionals, The Computer Security Institute found that 65% of companies represented had experienced a virus attack. Given the consistently expanding number of recorded malware and malicious threats, many vendors have stepped into the security space, offering anti-malware, anti-phishing, anti-spyware personal firewall and other security technologies seeking to deliver protection against these threats. The amount of security solutions and applications is in the thousands; every year vendors release additional solutions designed to provide faster and better protection against evolving threats, increase product usability, and support additional platforms.

The increasing quantity of security vendors and applications introduce new challenges to security vendors and system integrators tasked to classify, identify, manage and check currency of anti-malware applications -- whether the task is associated with an integration project to solve needs for specific customers, or building a new security solution that needs to interoperate with one or many anti-malware applications.

The Classification and Identity Challenges include a verification of anti-malware application binaries, especially when they could be compromised by malicious code. Malware can do that by creating binaries and executables with identical names, by adding similar registry keys or by reporting to the operating system.

These challenges extend with the existence of rogue applications – a rogue application is marketed as an anti-malware application. It reports to the operating system as an anti-malware application although it does not provide proven, reliable anti-malware protection. It may use unfair, deceptive, high pressure sales tactics to induce gullible, confused users to purchase.

The Manageability Challenge is a common programmatic way to control common features of anti-malware applications. Although each anti-malware vendor may offer similar functionalities such as scan or update, managing these functionalities programmatically differs from one solution to the other. Different anti-malware applications have different interfaces. Some anti-malware applications have an API program, others may have a well-documented CLI, but the interfaces are not consistent across vendors. For example – one vendor could expose definition update functionality, but others may not. Any integration attempt also faces interface quality aspects across the vendor spectrum as interfaces may break.

The Currency Checking Challenge - Many anti-malware applications are signature-based. This means that in order to keep the anti-malware application effective, it has to be current with the latest definition update. Many anti-malware vendors provide an update mechanism for their anti-malware engines. Vendor update mechanisms follow different security schedules such as hourly, daily, weekly or even monthly updates. This increases the complexity of currency checking because of the challenge of figuring out every update mechanism schedule and comparing it to the signature files on the local machines.

SECTION **2.0**

Object-Oriented Anti-malware Integrated Language

Object-Oriented Programming (OOP) is a great way to solve the anti-malware integration challenge. OOP is more than just a programming concept. It is a way of thinking about applications. It is learning to think of applications not as procedures, but as objects. Objects that do things (methods), and have attributes (properties), and are therefore logically grouped by the way they appear and behave.

If we'll perform an Object-Oriented design and analysis for the anti-malware application, the anti-malware application could be considered as the object, that object's methods could include: scanning files, scanning memory, triggering an update etc. The properties could include the name, version, language and type.

Abstraction is a powerful feature provided by object-oriented languages. The concept of abstraction relates to the idea of hiding data that is not needed for presentation, present only the information. The main idea behind data abstraction is to give a clear separation between properties of data type and the associated implementation details. This could be ideal to manage specific security application features as by hiding data or abstracting details that are not needed for presentation. For example: low level operation of an anti-malware could be hidden -- such as open or close a file while relevant logically methods could be exposed such as anti-malware.scan(); or anti-malware.update(). Other benefits of this abstraction is enhanced security - abstraction gives access to data or details that are needed by users and hides the implementation details, providing enhanced security for application. For example the method antimalware.clean(file) could be exposed while method like antimalware.cleanPrepare() or antimalware.cleanVerify() could be hidden.

Another feature of object-oriented programming is inheritance. Inheritance allows an object to have the same behavior as another object and extend or tailor that behavior to provide a special actions or special actions for specific needs.

Let's use the anti-malware application as an example. Both anti-malware applications "John" and "anti-malware Doe" objects have similar methods such as scanning a file and similar properties such as vendors and version names.

Rather than put these methods and properties in both of these objects, the method could be placed in a new object called object Anti-malware. Both anti-malware John and anti-malware Doe become child objects of the object Anti-malware, and both inherit the object Anti-malware's behavior.

SECTION **3.0**

Example of Pseudo Code Which Demonstrates Inheritance of Anti-malware Applications

```
Class CAnti-malware {
  Public:
        string name;
        version ver;
        date expirationdate;
        bool filesystem_protection_state;
        bool scanfile(CFile C) // code to clean file
        }
};

class JohnAnti-malware : public CAnti-malware
};
class DoeAnti-malware : public CAnti-malware
};

class CSecuritySuite : public CAnti-malware{
   private:
        bool anti-phishing_state ; // identifies whether the
anti-phishing is on
        }
};

The code to use this OOP could look like:

Begin program () {

JohnAnti-malware JohnAM;
DoeAnti-malware DoeAM;
CSecuritySuite ProAM;

display JohnAM.name();
display JohnAM.expirationdate();
display DoeAM.name();
display DoeAM. expirationdate();
display ProAM.name();
display ProAM. expirationdate ();

exit()
```

}JohnAnti-malware and DoeAnti-malware inherit CAnti-malware Object methods
CSecuritySuite inherits Anti-malware Object and adds additional Anti-phishing functions

SECTION **4.0**

Two Cases of Object-Oriented Anti-malware Integration Language

1. Remote Access

Remote access vendors are commonly challenged to assess the security health of endpoints which are the most vulnerable element in the network. The security health check includes verifying if the anti-malware application is installed, if it is authentic, if the security anti-malware file system feature is turned on, if the anti-malware application is up to date and, if the system was recently scanned and no malware were found.

Following object-oriented design principles, the final code could be as simple as:

```
Class CAnti-malware {
   Public:
       string name;
       version ver;
       bool isinstalled;
       date lastscantime;
       bool filesystem_protection_state;
       DefenitionFile def;
       Bool ishealthy(); // add code to define if the anti-malware
is authentic
       Bool isauthentic(); // add code to define health state
       bool scanfile(CFile C) // code to clean file
       }

Begin program () {

The OOP code could be as simple as:

   CAnti-malware DoeAM;
   display DoeAM.ishealthy();
   display DoeAM.isauthentic();

exit ()
};
```

2. Multi-scanning Solution

Problems with a single anti-malware engine approach stem from having only one system in place to identify threats. Although the signature files used by an engine to identify malware are generally updated several times a day, they are often released after a new malware has already hit and damage has been done. Even if an engine is 99.9 percent effective, it only takes one infection to inflict damage. Therefore, integrators and solution builders are seeking to implement a layered anti-malware approach, using a single, centrally managed solution that eliminates the need to evaluate different anti-malware scan engines and manage different vendors.

The following example demonstrates a simple multi-scanning solution:

```
int main() {

The OOP code could be as simple as:

   CAnti-malware JohnAM;
   CAnti-malware DoeAM;
   CFile file;
   JohnAM.scanfile(file);
   DoeAM.scanfile(file);

};
```

SECTION **5.0**

Conclusion

Object-oriented programming could provide an elegant easy to use solution to create a programmatic management layer to manage anti-malware and potentially other security solutions.

SECTION **6.0**

References

http://www.spywarewarrior.com/rogue_anti-spyware.htm
http://www.opswat.com/

SECTION **7.0**

About OPSWAT

OPSWAT is a global cyber security company that has provided security solutions for enterprises since 2002. Trusted by over 1,000 organizations worldwide, OPSWAT prevents data breaches and malware infections by eliminating security risks from data and devices coming into an organization. MetaDefender by OPSWAT is a powerful advanced threat detection and prevention platform, offering data sanitization (CDR), vulnerability assessment, multi-scanning, heuristics, big data, and additional threat protection technologies for a solution that is not solely based on detection. MetaAccess by OPSWAT is a cloud-based access control solution that helps enforce endpoint compliance and prevents contamination of cloud applications by blocking potentially compromised or noncompliant devices from accessing SaaS applications.

To learn more about OPSWAT's innovative and unique solutions, please visit http://www.opswat.com.