Security in Industrial Control Systems

The National Programme for Increased Security in Industrial Control Systems aims at enhancing the national capacity to handle Cyber-related threats to Industrial Control Systems (SCADA-systems) of critical importance to society. The programme's objectives are to increase the technical competence and to support users of such systems, in order to increase overall security. The Programme is run by the Swedish Civil Contingencies Agency in co-operation with the public and private sectors.

The Swedish Defence Research Agency, FOI, develops and runs the Programme's Technical Co-operation Platform. It consists of an advanced SCADA-laboratory with technical demonstrators, skill training courses in IT-security for SCADA operators, participation in national and international exercises, research collaboration and awareness raising activities.

FOI
Swedish Defence Research Agency
SE-164 90 Stockholm

Phone +46 8 555 030 00
Fax +46 8 555 031 00

www.foi.se

Swedish Civil Contingencies Agency
SE-651 81 Karlstad

Phone: +46 (0) 771-240 240
Fax: +46 (0) 10-240 56 00

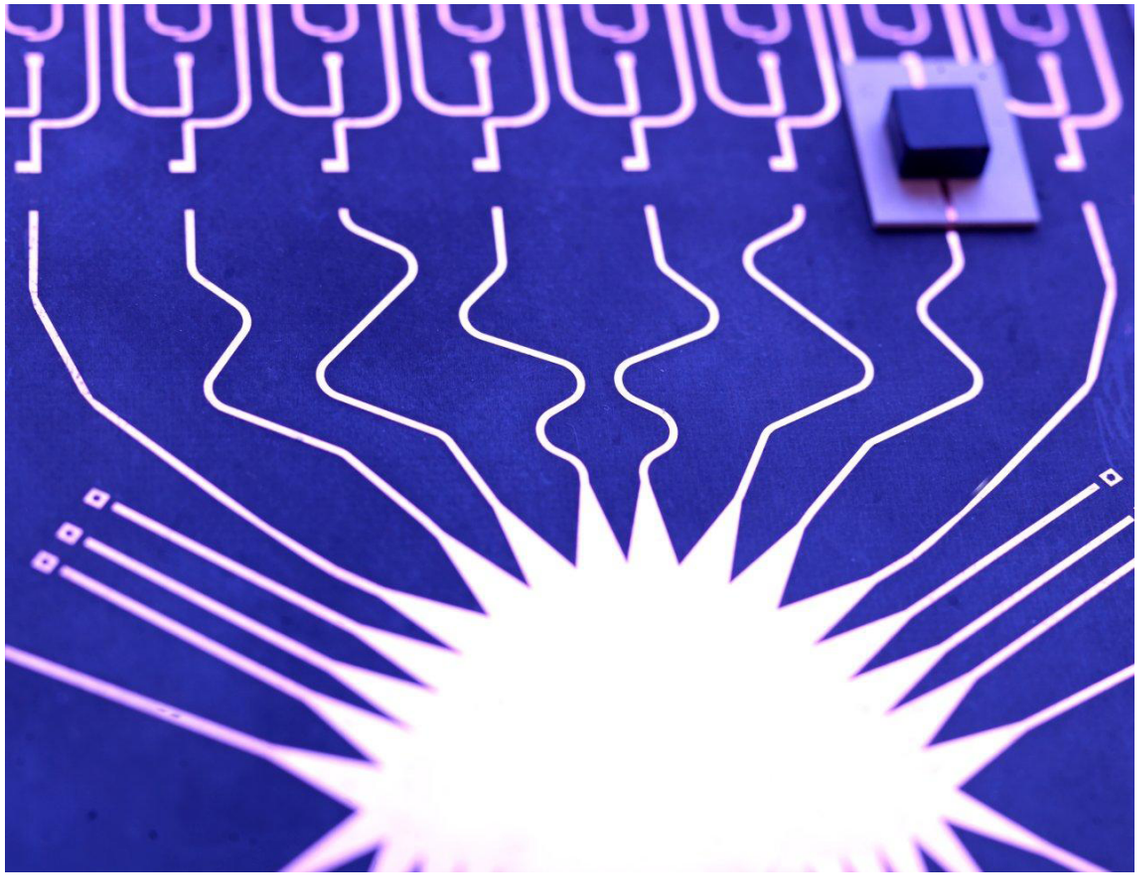
www.msb.se

Application whitelisting

Raises the bar against certain threats but no silver bullet

ARNE VIDSTRÖM

FOI
MSB



FOI-R--3434--SE
ISSN 1650-1942 April 2012

Arne Vidström

Application whitelisting

Raises the bar against certain threats but no silver bullet

FOI-R--3434--SE

Titel	Applikationsvitlistning
Title	Application Whitelisting
Rapportnr/Report no	
Månad/Month	April
Utgivningsår/Year	2012
Antal sidor/Pages	25 p
ISSN	1650-1942
Kund/Customer	MSB / Swedish Civil Contingencies Agency
FoT område	
Projektnr/Project no	E323125
Godkänd av/Approved by	Lars Höstbeck
Ansvarig avdelning	Informations- och aerosystem

Sammanfattning

Vitlistning har föreslagits som en lösning på några av de speciella säkerhetsproblem som finns hos SCADA-system (Supervisory Control and Data Acquisition). Av olika anledningar är sådana system svåra att uppdatera och det kan också vara problematiskt att köra och hålla antivirusprogram uppdaterade i dem.

Vitlistning handlar framför allt om att förhindra oavsiktlig exekvering av filer som kan innehålla skadlig kod. Man kan inte förvänta sig andra säkerhetsfunktioner från vitlistningsprodukter, som till exempel skydd mot buffertöverskridningsattacker, även om de kan innehålla sådana funktioner. Anledningen är helt enkelt att det ligger utanför vitlistningens uppgift. Ibland kan vitlistning i sig också skydda mot andra sorters attacker, men det är inget mer än en positiv bieffekt och inget man kan förlita sig på.

Det finns en konsensusförväntan om att allt som inte uttryckligen godkänns är spärrat vid vitlistning. Den här studien visar att det är en mycket förenklad bild av verkligheten. En konsekvens av det är att man kan förvänta sig att framtida skadlig kod ibland kommer innehålla funktionalitet som utnyttjar sårbarheter i vitlistningsprodukterna själva.

Vitlistning bör betraktas som ett användbart komplement till andra säkerhetslösningar. Skyddet som ges av vitlistning är inte tillräckligt för att ersätta uppdateringar av mjukvara och antivirusprogram. Vitlistning är ingen universallösning som kan ersätta andra säkerhetslösningar.

Nyckelord: vitlistning, SCADA, antivirus, skadlig kod, uppdatering

Summary

Whitelisting has been suggested as a solution to some of the special security problems faced by SCADA (Supervisory Control and Data Acquisition) systems. For various reasons, such systems can be hard to patch and it can also be problematic to run and keep antivirus software up-to-date on them.

Whitelisting is mainly about the protection against unintended execution of files which may contain malware. Specific whitelisting products may also contain other security features - for example protection against buffer overruns. Such features must not be expected though, since they are not part of whitelisting itself. Sometimes whitelisting itself will protect against other kinds of attacks too, but that is no more than a positive side-effect, and nothing to be relied upon.

There is also a consensus expectancy of default deny in whitelisting. However, this study shows that default deny is a very simplified picture of reality. A consequence of this is that we should expect future malware to sometimes contain circumvention functionality which exploits vulnerabilities in the whitelisting products themselves.

Whitelisting should be regarded as a useful complement to other security solutions. The kind of protection offered by whitelisting is not enough to replace software patching and antivirus software though. It is no silver bullet capable of replacing other security solutions.

Keywords: whitelisting, SCADA, antivirus, malware, patching

Contents

1 Introduction 7

2 The security model of whitelisting 8

3 Security features are not the same as secure features 11

4 The true complexity is revealed if we look under the hood 13

5 The monitoring of native executable files is illustrative 14

5.1 Default deny is easier said than done 15

5.2 Extensive error handling can break default deny too 15

5.3 What does not seem to matter can matter too 16

6 How to evaluate whitelisting products 18

6.1 How to evaluate coverage 19

6.2 How to evaluate implementation robustness 20

6.3 The risk of a system crash should not be forgotten 21

7 Malware trends and whitelisting products 23

8 Conclusions 25

1 Introduction

The interest in application whitelisting is growing.[1] In particular, whitelisting has been suggested as a solution to some of the special security problems faced by SCADA (Supervisory Control and Data Acquisition) systems.[2] For various reasons such systems can be hard to patch and it can also be problematic to run and keep antivirus software up-to-date on them. There has been a lot less focus on how much one can trust whitelisting products, although there are recent examples of these kinds of evaluations.[3] [4] However, these examples are mainly targeted against aspects where there is no consensus about what to expect from the products.

The purpose of this study has been to evaluate how useful whitelisting is for SCADA purposes – in particular if it can solve the special security problems faced by these systems.

The study has been financed by the Swedish Civil Contingencies Agency (MSB) as part of the cooperation NCS3 between MSB and the Swedish Defence Research Agency (FOI). It has also involved the industrial partners Siemens Industrial Turbomachinery (SIT) and ABB Automation Technologies.

[1] N. McDonald, Gartner, *Application Control / Whitelisting Interest is Growing Rapidly*, 2010-05-11, accessed 2012-04-02, <http://blogs.gartner.com/neil_macdonald/2010/05/11/application-control-whitelisting-interest-is-growing-rapidly>

[2] M. Hines, eWeek, *Apps Whitelisting Proponents Tout Growing Acceptance*, 2009-06-25, accessed 2012-04-12, <http://securitywatch.eweek.com/applications_whitelisting/apps_whitelisting_proponents_tout_growing_acceptance.html>

[3] Foreground Security, *Raising the White Flag - Bypassing Application White Listing*, 2012-02-02, accessed 2012-04-12, <http://www.foregroundsecurity.com/blog/raising-the-white-flag-bypassing-application-white-listing.html>

[4] D. Peterson, Digital Bond, *2 x S4 Videos on Application Whitelisting in ICS*, 2012-02-02, accessed 2012-04-12, <http://www.digitalbond.com/2012/02/02/2-x-s4-videos-on-application-whitelisting-in-ics>

2 The security model of whitelisting

The concept "security model" is usually used to refer to formal theoretical security models like the Bell–LaPadula model (BLP), multilevel security (MLS), or role-based access control (RBAC). In fact there is some kind of security model behind most software products, but it might just not be that explicit. We assume some things about the products security-wise and we do not demand other things from them.

When we encounter a new class of software there might be some initial confusion about its security model if it is not stated explicitly. For example, the whitelisting product Application Control from McAfee offers some amount of protection against buffer overruns.[5] Is whitelisting in general supposed to block the execution of unwanted code in any form? Some people seem to think so, but most do not. Here are a few examples where the central task of whitelisting is stated explicitly:

- "Most application-control vendors control whether a given file can be executed or not".[6]
- "It only allows certain trusted files to run on your machine".[7]
- "Whitelisting involves barring all but approved executables from running on a given machine".[8]

The consensus implicit security model of whitelisting is all about blocking the execution of files which are not in the whitelist database. In addition, some of the products have extra features which have nothing to do with whitelisting. For example, in a 2009 review by InfoWorld, two of the six tested products offered some amount of protection against buffer overruns.[9] It is not the task of whitelisting itself to protect from attacks against vulnerabilities in whitelisted executable files, which is the case with for example buffer overruns. However,

[5] McAfee, *McAfee Application Control*, accessed 2012-04-02, <http://www.mcafee.com/us/products/application-control.aspx>

[6] N. McDonald, Gartner, *Application Control / Whitelisting Interest is Growing Rapidly*, 2010-05-11, accessed 2012-04-02, <http://blogs.gartner.com/neil_macdonald/2010/05/11/application-control-whitelisting-interest-is-growing-rapidly>

[7] R. Vamosi, CNET, *Column: Will you be ditching your antivirus app anytime soon?*, 2008-07-21, accessed 2012-04-02, <http://news.cnet.com/8301-10789_3-9994679-57.html>

[8] J. Brooks, eWeek, *Application Whitelisting Gains Traction*, 2008-09-25, accessed 2012-04-02, <http://www.eweek.com/c/a/Security/Application-Whitelisting-Gains-Traction>

[9] R. Grimes, InfoWorld, *Whitelisting security solutions by the features*, 2009-11-04, accessed 2012-04-02, <http://www.infoworld.com/node/98873>

there have been evaluations of whitelisting products where buffer overrun protection was included in the tests anyway.[10]

Not even all executable files are monitored by the whitelisting products though. This is the question of *coverage*. One must not automatically expect that for example Java, ActiveX or kernel modules are monitored.[11] There are also special problems with various more or less obscure interpreted languages, since it cannot be expected that whitelisting products should be able to cover them all. The question of coverage has been a central point in at least one evaluation of whitelisting products.[12]

Another question is the extent to which the products should protect against the actions of ordinary users. The main purpose of whitelisting is to protect against users trying to execute files containing malware. But should such products protect themselves against a user who might try to circumvent them on purpose or who might break them through some trivial mistaken action? There is no consensus about the answer to this question. There are differences between the products, both regarding if such protection is included at all, and regarding how inclusive such protection is when it is included.[13] For example, Application Control from McAfee protects all its components from being deleted or renamed. On the other hand, SE46 from Cryptzone does not protect vital components from being renamed. Simply renaming a vital component can potentially render a whitelisting product installation useless. Such an attack against SE46 will be presented later.

[10] D. Peterson, Digital Bond, *2 x S4 Videos on Application Whitelisting in ICS*, 2012-02-02, accessed 2012-04-12, <http://www.digitalbond.com/2012/02/02/2-x-s4-videos-on-application-whitelisting-in-ics>

[11] D. Shackleford , SANS, *Application Whitelisting: Enhancing Host Security* , 2009-10, accessed 2012-04-02, <http://www.sans.org/reading_room/analysts_program/McAfee_09_App_Whitelisting.pdf>

[12] Foreground Security, *Raising the White Flag - Bypassing Application White Listing*, 2012-02-02, accessed 2012-04-12, <http://www.foregroundsecurity.com/blog/raising-the-white-flag-bypassing-application-white-listing.html>

[13] D. Peterson, Digital Bond, *2 x S4 Videos on Application Whitelisting in ICS*, 2012-02-02, accessed 2012-04-12, <http://www.digitalbond.com/2012/02/02/2-x-s4-videos-on-application-whitelisting-in-ics>

Summary

At this point we can put together at least a sketchy picture of the implicit security model of whitelisting. Its central task is to protect the system from the unintended execution of files that may contain malware. There is a consensus expectancy of default deny. There is however no consensus about the exact coverage of file types to be monitored. We cannot expect any other security features from whitelisting products, like protection against buffer overruns, although some products may contain certain extra features. Finally, there is no consensus about how well the products should protect themselves against the actions of users.

3 Security features are not the same as secure features

The previously mentioned InfoWorld whitelisting review from 2009 is pretty representative of many reviews of security products in general. They often focus on things like the graphical user interface, ease of configuration, and similar. Of course those aspects are important, but the problem with this kind of review is illustrated nicely by the comic in Figure 1.

Figure 1. TornadoGuard App Review by Randall Munroe.[14]

The reason why you choose to run a security product in the first place is probably that you wish to protect yourself against various threats. Unfortunately, most reviews fail to measure that capability almost completely. It is much more

[14] <http://xkcd.com/license.html>

complicated to rate the security of the features of a product than to rate the number of and general appearance of its security features.

4 The true complexity is revealed if we look under the hood

To gain an understanding of just how complex the question of the reliability of whitelisting products is, we have to look under the hood of them. Most people probably have the impression that whitelisting entails scarce more than a software module that calculates some kind of checksum for each executed file and compares it to a database of whitelisted files.

In reality, the whitelisting products have a much larger number of tasks which they go through before they perform the final comparison between the file and the whitelist database. First of all, they do not just monitor one single point in the operating system and check everything that passes through it. Instead they monitor various checkpoints, and different whitelisting products utilize different techniques for that task. Next, they do not simply do a comparison for every single file they discover at such a checkpoint. Instead they first perform various tests, and then decide if it is appropriate to make a comparison with the whitelist database at all. Here we will focus on a single type of file for illustrative purposes, but remember that there are other types as well. Also remember that the general impression of whitelisting is that it is all about default deny, but as you will see there is a certain amount of default allow at a level below the default deny.

5 The monitoring of native executable files is illustrative

The file type we will focus on is the native executable file. These are the ones that contain x86 machine code and are run directly by the user. The most widely known example is probably the EXE file in Windows. This should really be the strong suit of the whitelisting products. Not keeping track of obscure interpreted languages is one thing, but keeping track of native executable files is spot on in the middle of their role.

We will begin by taking a look at the file header of a modern EXE file. In fact, a modern Windows EXE file starts with a file header that is similar to a legacy EXE header from MS-DOS 2.0 (anno 1982). The reason is backwards compatibility. The legacy header contains 14 different items, the first of which is a signature consisting of the letters 'MZ'.[15] The other 13 items specify things like the size of the file, memory relocation information, the size of the header itself, initial values for different processor registers in 16 bit mode, and so on.

Already at this point we have a lot of combinations of items and values to play tricks with. For example, we can construct a file that states that its header is non-existent (zero length) but indeed does have a header. There is also a checksum value in the legacy header, and we can set it to a proper value or to a faulty value. These are just a couple of examples.

What happens when we play these tricks depends on the one hand on how Windows interprets the values and on the other hand on how the whitelisting products interpret them. If they both ignore the values completely it does not matter what we set them to. If Windows accepts them while the whitelisting products do not accept them there *might* be a problem. Either the file will be default denied or it might be default accepted depending on the implementation of each whitelisting product.

However, a modern EXE file does not end with the legacy header. The true header of such a file is called a PE header (Portable Executable). A bit further into the file from the legacy header, at position 3Ch (hexadecimal), we can find an item that specifies the location of the PE header.[16] The PE header itself contains much more information than the legacy header does. In order not to digress too much from the main topic we will not look at the contents of the PE header itself. Instead we will work through the details of two actual

[15] Microsoft, *Microsoft MS-DOS Programmer's Reference Version 5*, Microsoft Press, Redmond, 1991, p. 76.
[16] Microsoft, *Microsoft Portable Executable and Common Object File Format Specification*, 2010, p. 11.

vulnerabilities in whitelisting products. The vendors have been notified about both vulnerabilities and patches have already been released a few months ahead of the publication of this report.

5.1 Default deny is easier said than done

The first example comes from the product SE46 by Cryptzone.[17] Remember that an EXE file starts with the letters 'MZ'. The first thing SE46 does is to look at these two bytes. If they are present it goes on to check for other things, like the PE header. If they are not present it takes a look at the extension of the file. If the extension is not BAT, CMD, COM or EXE it lets the file through for execution by Windows.

How can we take advantage of this implementation? One way is if we have a file with native x86 code that does not start with the letters 'MZ' and does not have one of the four extensions. There is a kind of executable file that does not start with 'MZ' since it is completely headerless. It is the 16-bit COM file from MS-DOS. We can assemble any 16-bit COM file we wish and try to execute it. It will pass through the file header check, but it will be caught at the file extension check. All we need to do now is find an extension that Windows accepts as executable but which is not in the SE46 shortlist. PIF (Program Information File) is such an extension. Now we can run any 16-bit COM file just as long as we make sure it has a PIF extension.

5.2 Extensive error handling can break default deny too

The second example comes from the product Application Control by McAfee.[18] A closer look at this product reveals that it appears to have more extensive error handling than SE46. At least in the parts we are concerned with here. Extensive error handling is usually very good, but sometimes it can be turned against you, too. Application Control first of all checks if the file has a valid header, including the PE header, or not. If the header is not valid the file is let through to Windows for execution, presumably because it is assumed to be non-executable.

[17] Cryptzone, *SE46 Application Whitelisting*, accessed 2012-04-03, <http://www.cryptzone.com/products/se46-application-whitelisting>

[18] McAfee, *McAfee Application Control*, accessed 2012-04-03, <http://www.mcafee.com/us/products/application-control.aspx>

This time, remember that the location of the PE header is specified at position 3Ch (hexadecimal) in an EXE file. The location is specified by a four byte long value, so that the positions 3Ch, 3Dh, 3Eh and 3Fh are used for this purpose. Now we create a file that is so small that it is missing the position 3Fh. This means that the file is really too small to be a PE-style file.

Next, we set the legacy EXE header size to zero. Now it looks like the file is missing a legacy EXE header too. However, there is still the problem that the file ends with the extension EXE, so instead we change it to SCR (screensaver).

If we execute such a file in Windows, without Application Control installed, it will execute despite all these problems. It will look like a legacy EXE file to Windows, and the execution will start from the very beginning of the file, with the letters 'MZ'. Fortunately these letters are in fact executable too, because when interpreted as machine code instead of as text, they mean something at least remotely comprehensible to the CPU. Next, the execution continues with the other items in the header. If we pick the right values for these we can make sure that the values are on the one hand executable, and on the other hand mean something as header values too. We also have more space following this, to the end of the file, where we can insert any code we like without worrying about what it means value-vise.

Now we have a file which executes as a 16-bit legacy EXE file in Windows. If we run it in a system protected by Application Control it will execute whether it is in the whitelist database or not. Application Control finds the file invalid and simply passes it on to Windows for execution.

5.3 What does not seem to matter can matter too

In both examples the file we managed to execute contains 16-bit code. This kind of code has its limitations in Windows compared to ordinary 32-bit or 64-bit code. For example, it cannot make Win32 API calls. One thing it can do though is rename files. Remember that the security model of SE46 does not offer protection against the actions of the users. As soon as our specially crafted executable starts to execute it looks just like an ordinary user to the system. It is limited by Windows only because it is 16-bit, but SE46 no longer takes responsibility for protecting against its further actions. Thus, we can let our executable rename a few core components of SE46 if we are running on a sufficiently privileged account. After the next reboot SE46 is no longer running on the computer. If we try to do the same thing against Application Control we will not succeed, since it protects its core components from the actions of the

users. What might not seem to matter for whitelisting purposes in fact turns out to matter a great deal.

Summary

There is a consensus expectancy of default deny in whitelisting, but as we have seen, that is a very simplified picture of reality. In fact, it is fair to say that there is default allow in the foundation of whitelisting. The default state of a computer system is that all kinds of files can be executed through various paths. Whitelisting products must keep track of all of these paths, keep track of what tries to go through them, and make a lot of decisions in the process. Only when the right set of conditions is present, an executable is default denied and subjected to a final test to see if it should be allowed to execute or not. When any other set of conditions is present, it is default allowed to execute.

Finally, there is no consensus about how well the products should protect themselves against the actions of users. One implication of this is that some products are more vulnerable than others to the consequences of limited penetration of their file monitoring.

6 How to evaluate whitelisting products

As we have seen so far, there are at least three aspects of whitelisting products which we have to take into account if we wish to evaluate them:

- Usability
- Coverage
- Implementation robustness

Many reviews of security software are unfortunately restricted to evaluations of usability. The previously mentioned InfoWorld review also looked at some aspects of coverage though, but not at all at implementation robustness. This problem is not limited to reviews in computer magazines and the like. For example, the Information Security Management Handbook has a section called Evaluating Whitelisting Products.[19] It lists a number of attributes of whitelisting products which should be taken into consideration when evaluating them against the requirements one has:

- Manageability
- Deployment
- Policy definition
- The end-user experience

Readers who wish to know more about how to evaluate manageability, deployment, policy definition and the end-user experience should turn to the Information Security Management Handbook or a number of other sources which have more to say about these issues.[20] [21] [22] [23] [24] The vendors also offer quite a lot

[19] R. Shein, 'Whitelisting for Endpoint Defense', in *Information Security Management Handbook Volume 5*, M. Krause Nozaki & H. Tipton (eds), CRC Press, Boca Raton, 2012, pp. 11-12.

[20] R. Grimes, InfoWorld, *InfoWorld review: Whitelisting security offers salvation*, 2009-11-04, accessed 2012-04-03, <http://www.infoworld.com/d/security-central/test-center-review-whitelisting-security-offers-salvation-835?page=0,0>

[21] D. Shackleford , SANS, *Application Whitelisting: Enhancing Host Security* , 2009-10, accessed 2012-04-03, <http://www.sans.org/reading_room/analysts_program/McAfee_09_App_Whitelisting.pdf>

[22] NSA, *Application Whitelisting*, accessed 2012-04-03, <http://www.nsa.gov/ia/_files/factsheets/Application_Whitelisting_Trifold.pdf>

[23] S. Bisson, IT Expert Magazine, *Application Whitelists*, 2010-05-11, accessed 2012-04-03, <http://www.itexpertmag.com/security/application-whitelists>

[24] R. Abrams, ESET, *White Listing - The End of Antivirus?*, 2008-11-16, accessed 2012-04-03, <http://blog.eset.com/2008/11/16/white-listing-%E2%80%93-the-end-of-antivirus>

of information about such things for marketing purposes. They are of course very important issues in an evaluation, but here we shall focus on issues which are all too often forgotten about.

6.1 How to evaluate coverage

Evaluating coverage consists of two parts: investigating which executable file types are covered by a specific product and investigating the total number of executable file types that exist for a particular system. With that information it is easy to calculate the coverage as a percentage for that product on that system.

Investigating which file types are covered by a specific product can be done in different ways. Some of the information can be found in the marketing and technical documentation. Exactly how detailed such sources are varies from vendor to vendor. It can be complemented by contacting the vendor and asking for further information. This kind of evaluation should be done as a minimum. It is even better to retrieve an evaluation license for the product in question and test that the stated information is indeed correct. Such tests are not technically advanced and can be performed by a normal testing department without special knowledge of security.

The other part of a coverage evaluation is much harder. The hard problem is how to make sure that all executable file types in a system have been identified. First of all there are the well-known file types of the particular operating system, then the less well-known, then interpreted languages (like for example Perl) which may not be default on the platform, and finally proprietary interpreted languages and similar. Advanced software products often have some kind of proprietary language built in. The exact number of executable file types varies from system to system depending on the software combinations installed.

The coverage quotient depends on two factors: the whitelisting product and the system to be protected. Its value can be raised either by using a better whitelisting product, or by decreasing the number of executable file types in the system. Different whitelisting products may have different coverage quotients for different systems depending on the combinations of executable file types they handle.

6.2 How to evaluate implementation robustness

Evaluating implementation robustness is a much more technically advanced task than evaluating coverage. Robustness is a measure that depends on how the product is implemented in detail. Very small details can make all the difference, as we have seen in the earlier examples. And there are indeed a lot of these small details in a whitelisting implementation. As figure 2 illustrates, there are many parallel paths which must to be investigated in order to completely determine robustness, and many individual steps in each path.

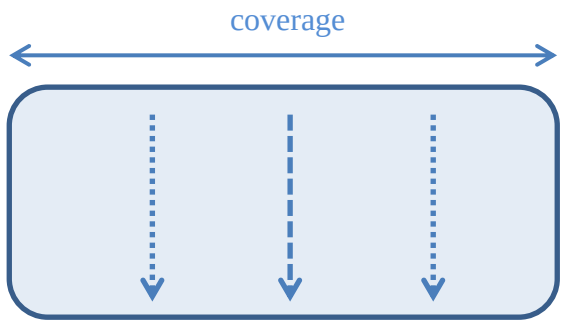


Figure 2. Coverage versus implementation robustness.

For example, one main path could be Java file monitoring and another could be EXE file monitoring. Each main path consists of several sub-paths. We have seen that there are a number of checks done when implementing EXE file monitoring, and still we have only scratched the surface.

In figure 2, the arrow in the middle represents our investigations into EXE file monitoring. It is medium dashed instead of solid to represent the fact that we have not looked at every single detail. Instead we have sampled a few specific details and found a couple of vulnerabilities. The dotted arrows on the sides represent all the other paths that we have not even taken a cursory look at.

There are probably few organizations which can afford even something remotely close to a complete evaluation of a whitelisting product. The total number of implementation details will be huge indeed, their combinations even more numerous, and without access to the source code the work will be monumental.

A better way is to perform a number of spot checks down a selected path, or down a few selected paths. Combined with our knowledge about coverage we

will at least be able to compare two products against each other with some level of certainty.

Unfortunately this kind of testing cannot be done by most testing departments because it takes a very specialized skill set. If it is found important enough the product must probably be sent to an independent security lab for testing.

If such detailed testing is not possible, for financial or other reasons, a simpler level of evaluation is a reasonable alternative. At a minimum, the vendor should have an internal code review process for their product. There should be at least some person who is not involved in the actual development who performs code security reviews. If there is no such process in place there is a very high risk that the robustness of the product is lacking.

There are also security aspects of whitelisting products which have nothing to do with the file monitoring itself. They include but are not limited to:[25]

- Attacks against management functions
- Attacks against software distribution points
- Stolen software certificates
- Attacks against the whitelisting database
- Malicious insiders
- Attacks against administration accounts

Some of these demand further evaluation of the whitelisting products, and some demand further security actions inside the computer network of the organization where the product is to be deployed.

6.3 The risk of a system crash should not be forgotten

Another technical aspect of evaluating a whitelisting product is ensuring that it does not crash the system. Whitelisting products may access various undocumented parts of the operating system, and they usually contain kernel mode modules. Thus, there is a potential risk of a system crash, including the infamous blue screen of death in Windows, because of a software bug.

A crash may be one thing in an office workstation and another thing in a critical system like for example a SCADA system. The problem is especially precarious

[25] J. Beechey, SANS, *Application Whitelisting: Panacea or Propaganda?*, 2010-12, accessed 2012-04-03, <http://www.sans.org/reading_room/whitepapers/application/application-whitelisting-panacea-propaganda_33599>

when the vendor of the whitelisting product is small and has a small user base. Then the vendor might not have performed enough testing and there may be no other users who have run the particular system set-up in question. Kernel modules can become instable for a number of reasons and in a number of situations, for example depending on the system load. Interactions between different kernel modules can also lead to crashes. Therefore it is vital to perform extensive testing before deployment in a critical system. Small vendors may also be unable to quickly identify the cause and offer a solution if a system crash indeed occurs.

Summary

The evaluation of whitelisting products is unfortunately often limited to factors like manageability, deployment, the end-user experience, et cetera. For security reasons it is also very important to evaluate coverage, robustness, and so on. Coverage can be evaluated fairly well by a normal testing department. The robustness of the implementation can be evaluated by sending the product to an independent security lab where the specialized skill set needed is available. A realistic ambition is to have the lab do some spot checking at various points in the product. Then different products can be compared to each other with at least some amount of certainty. At a simpler level, a product should at least be evaluated by checking that the vendor has a satisfactory internal code review process in place.

There are other important security factors too which have nothing to do with the file monitoring itself. They should also be evaluated, but the details are beyond the scope of this report.

7 Malware trends and whitelisting products

As we have seen, there may be vulnerabilities in the whitelisting products themselves. Such vulnerabilities will most likely be discovered and published in limited numbers in the future. It is not trivial to find a new vulnerability on demand when needed, but on the other hand it should be expected that such discoveries will be made every now and then. More vulnerabilities will be found the more popular whitelisting becomes and the more interest it generates among security researchers and hackers.

If whitelisting becomes popular enough, we should expect some malware to implement circumvention functionality by default. However, it is unlikely that the majority of malware will contain such functionality unless the use of whitelisting becomes very widespread. Whitelisting will probably offer good protection against the absolute majority of file-based malware in the future too.

When malware with circumvention code starts to spread, ordinary antivirus software will probably handle the threat much more quickly than the vendors of whitelisting products will be able to offer patches for their products. Antivirus vendors can usually put a new signature in their database within a day or so when a new virus appears in the wild.[26] At least that applies as long as the antivirus engine itself is not attacked. Whitelisting vendors, on the other hand, will always be delayed by a comparatively slow process of verifying the vulnerability, designing and implementing a solution, and finally more or less extensive testing. The average time from public disclosure to patch is 28 days, and 63 days from notification to the vendor only.[27] These numbers apply for software in general, but there is no reason to assume a significantly quicker response time from whitelisting vendors. For example, the times from notification to vendor to patch release for the two vulnerabilities found in this study were 19 days, 28 days, and 53 days.[28]

Whitelisting will also not protect as well against sophisticated customized attacks against a specific target, since such an attack can be custom-made to circumvent

[26] B. Livingston, eSecurity Planet, *How Long Must You Wait for an Anti-Virus Fix?*, 2004-02-23, accessed 2012-04-16, <http://www.esecurityplanet.com/views/article.php/3316511/How-Long-Must-You-Wait-for-an-AntiVirus-Fix.htm>

[27] A. Arora, R. Krishnan, R. Telang, & Y. Yang. 'An Empirical Analysis of Software Vendors' Patch Release Behavior: Impact of Vulnerability Disclosure'. *Information Systems Research*, vol. 21, March 2010, pp. 115-132.

[28] One of the products needed two patches, for different versions of it.

the particular whitelisting product used by the target. In such a case the vulnerability used will probably be a zero-day.

On the other hand, whitelisting will raise the bar considerably against most local file-based attacks, including the ones listed above.

Summary

In the future we can expect malware that targets vulnerabilities in the whitelisting products themselves to circumvent their protection. Most likely only a small part of all malware will do so, but in such cases ordinary antivirus software is likely to offer protection against the threat much quicker than the whitelisting vendors. We can also expect a small number of sophisticated customized attacks against specific targets where the protection offered by whitelisting products will be circumvented through zero-day vulnerabilities. All in all, whitelisting will at least raise the bar against most file-based attacks.

8 Conclusions

For various reasons SCADA systems can be hard to patch and it can also be problematic to run and keep antivirus software up-to-date on them. As we have seen, whitelisting is limited to protecting against certain file-based threats. That kind of protection is of course not enough to replace software patching. Neither does it cover all that antivirus software can do. For example, antivirus vendors can act much more quickly than whitelisting vendors against new malware that circumvents whitelisting products. Whitelisting is also roughly as vulnerable against customized attacks as many other security solutions are.

Whitelisting should be regarded as a useful complement to other security solutions, including antivirus software. It raises the bar against certain kinds of attacks, and it can sometimes protect against other kinds of attacks as a positive side-effect. However, it is no silver bullet that replaces any other security solution.