INSECURITY IN SECURITY SOFTWARE

Maik Morgenstern
AV-Test GmbH, Klewitzstr. 7, 39112 Magdeburg, Germany
Tel +49 391 6075464 • Email mmorgen@av-test.de

Contributors:

Andreas Marx
AV-Test GmbH, Klewitzstr. 7, 39112 Magdeburg, Germany
Tel +49 391 6075466 • Email amarx@av-test.de

Mary Landesman
Antivirus.About.com, About, Inc., 249 West 17th Street, New York, NY 10011, USA
Email antivirus.guide@about.com

ABSTRACT

Data security software and, in particular, AV programs are deployed widely throughout companies, organizations, and private homes. Without this protection, users are at high risk of malware infection. But what happens when the protective software becomes the vector for compromise? In the first part of 2005, several security vulnerabilities – especially buffer overflows – were discovered in a wide range of security products. Both open source software such as *ClamAV* and commercial tools from *Symantec*, *F-Secure*, *Trend Micro* and *Computer Associates* have been affected. In this paper, we discuss the additional risk of infection caused by these vulnerabilities in AV and other security software, including how this risk can be reduced by the developers and by the users of the products.

1. A BRIEF INTRODUCTION

As mentioned in the abstract, this paper will deal with security software and the security vulnerabilities that may be found in them. Therefore, it is necessary to define what we mean by these terms. These should not be taken as general definitions, but rather as working definitions for use in this paper.

First we will define briefly our views on security software, then on security in general and, finally, we will take a brief look at the term 'security vulnerability'. However, we will not go into the details of security models or the basics of software engineering that will be needed to understand some of the reasons for security vulnerabilities presented later. We assume a very general knowledge of these issues.

1.1. Security software

Security software products can be divided into two groups. On the one hand, there are applications meant for the home and business user and, on the other hand, the tools used by researchers in the field of computer security. The first category contains products such as firewalls, IPsec products, network intrusion prevention/detection software, as well as cryptography applications and last but not least anti-virus (AV) software. The second category refers to tools like *IDA Pro*, *OllyDbg* and *Softice* or more generally debuggers, disassemblers, hex editors and the like.

This is by no means an exhaustive list and probably more or different categories could be included. But that will not be necessary since we are focusing on exactly these two kinds of security software, especially anti-virus, in the present paper. The reasons for this will be presented in section 2.

1.2. Security

At least a short introduction to the terms of security is needed, especially since some of the issues involved are very useful in the following parts of the paper.

> 'Security is about well-being (integrity) and about protecting property or interests from intrusions, stealing or wire-tapping (privacy – the right to keep a secret can also be stolen). In order to do that, in a hostile environment, we need to restrict access to our assets. To grant access to a few, we need to know whom we can trust and we need to verify the credentials (authenticate) of those we allow to come near us.' [1]

Security is based on the following independent issues:

- Privacy
- Trust
- Authenticity
- Integrity

Environments can be hostile because of:

- Physical threats
- Human threats
- Software threats – this is what we'll look at.

What are we afraid of?

- Losing the ability to use the system.
- Losing important data or files
- Losing face/reputation
- Losing money
- Spreading private information about people [1].

As one can recognize, most of those points will get your attention immediately, since they somehow seem to focus on the problems and issues we get with security holes in security software. But note: these are general thoughts about security in the computer field.

I want to close the definition with yet another quote, which introduces a problem we will be facing more often throughout the paper:

> 'Security is a social problem, because it has no meaning until a person defines what it means to them' [1].

1.3. Security vulnerability

It's pretty difficult to define the term 'security vulnerability' in the context of security software, since every simple bug or design error could possibly lead to a security issue without actually fitting into a common definition. Also, not everyone has the same idea of security, as stated above. Therefore we will first explain 'security vulnerability' in a general sense and later mention what additional problems might occur with security software.

On wikipedia.org the term ‘security vulnerability’ is explained as follows:

‘In computer software, a security vulnerability is a software bug that can be used deliberately to violate security’ [2].

However, security vulnerabilities don’t always depend on bugs. Simple misconfigurations/usage or even errors in design could lead to security flaws as well. Therefore, a more sophisticated definition is needed. The following can be found in *Introduction to Computer Security* by Matt Bishop [3]:

‘When someone breaks into a computer system, that person takes advantage of lapses in procedures, technology, or management (or some combination of these factors), allowing unauthorized access or actions. The specific failure of the controls is called a vulnerability or security flaw; using that failure to violate the site security policy is called exploiting the vulnerability.’ [3]

This is quite a sufficient definition. However, it still leaves some room for interpretation. Think of security software that releases a flawed update, rendering the systems the software is installed upon inoperable. One must either remove the security software or shut down the system. The first option has an impact on security, the second on business. Other problems that lead to denial of service or just bypassing viruses in a scan need to be considered when talking about security flaws. That’s why we’ll use a somewhat fuzzy idea of ‘security vulnerability’ throughout the paper.

2. WHY ARE WE LOOKING AT IT?

Now that we know what we are looking at, it’s to time to understand why we are looking at it. First, there is a general reason. More and more security vulnerabilities are being found and, more importantly, exploited in today’s software. That’s why security holes are still an up-to-date topic. According to the *Yankee Group*, 77 vulnerabilities in security products were disclosed from 2004 to May 2005 [4].

Of course this is a problem for the software industry in general and doesn’t affect security software alone. Why would we want to focus on security software? There are two main points that must be reviewed. The rate of incidents found in security software has increased a lot in the past months, even more than for *Microsoft*. Secondly, and in conjunction with the aforementioned aspect, security products such as anti-virus software are very wide deployed throughout homes and businesses and therefore are a *nice target* for vulnerabilities and exploits – not just hackers but also competitors or at least security analysts.

Let’s look at these two facts and reasons in a bit more detail. *Yankee Group* again gives a nice review on these points. *Microsoft Windows* has long been the number 1 target of vulnerability hunting (see Figure 1), but with its increased security efforts, the number of vulnerabilities found is decreasing and new targets are needed. And it seems that security software is a nice target for that purpose, because basically every business and home user has some security software installed. Another article confirms this: the *SANS* Top 20 Internet Security Vulnerabilities list currently contains well-known security vendor names like *Symantec*, *Trend Micro*, *McAfee* and *F-Secure* [5].

Additionally, there isn’t much public awareness, or even pressure, regarding security holes in security software. As

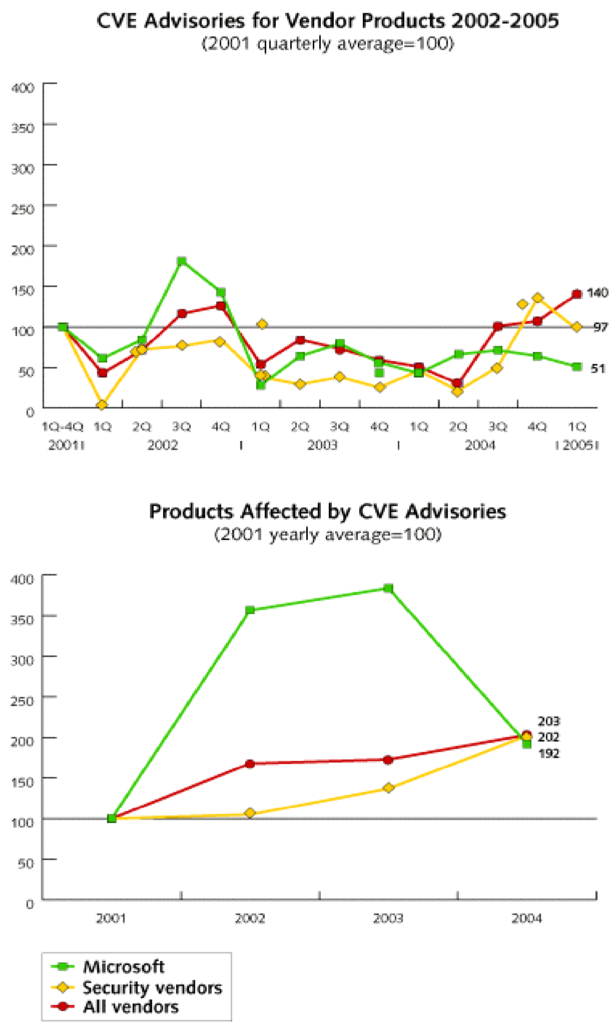

Figure 1 (source: [4]).

such, vendors aren’t really forced to react and fix or even redesign their software immediately – unlike *Microsoft* which is forced to do so, since it receives a lot of criticism when it doesn’t.

What happens when vulnerabilities are found and exploited was shown in early 2004 with the Witty worm, made possible solely through a bug in security software by *ISS*. *ISS* did, however, react swiftly and the number of vulnerabilities found in its software has been reduced since then. Other vendors don’t seem to have reacted just yet, thus inviting ‘black hat’ as well as ‘white hat’ hackers to exploit or find holes in their software. While the black hat side has always been a big part of the game, the white hat side will play an increasing role in the future. Security vendors, or at least security analysts, might check competitive products for vulnerabilities and release advisories, thus giving them bad publicity.

This leaves us with a paradox: security software is meant to secure the system, but nowadays it introduces new security holes. This is likely to impact on the vendors’ sales figures sooner or later, since users will likely prefer a more secure system – one that works correctly (at least most of the time) and doesn’t introduce new security holes or put the user at risk.

It becomes even more obvious when we remember some of the terms involved with security. People have certain expectations of software as well as security. *Microsoft*’s trustworthy computing paper of gives a good overview. Without going into detail we can sum it up as follows: your users need to trust your software to consider it to help secure their system; and the other way around, your software needs to be secure (and not just that, but that's the point we are reviewing here) to be trusted by the user [6].

Now it's time to present some examples that show some of the current dangers clearly. This will lay the foundation to look at some of the causes for these errors and vulnerabilities, which will lead to the insight: it's time to act! We will then try to give some hints for both the vendors and users on how to act and react to the current situation.

3. EXAMPLES AND THEIR EFFECTS

Now after all the theory it's time for some examples, since nothing gives a better view of a problem than a real-life experience. It is also necessary to understand the spectrum of security flaws and their reasons.

Just as we had problems defining the term security vulnerability, we will have problems with classifying these examples. When looking at them, one could come up with these classes:

- A bug that leads to a classic security vulnerability like a buffer overflow, which could be used to execute arbitrary code.
- Something other than a bug, but still leading to a security vulnerability or a bug that leads to unwanted behaviour, but which isn't necessarily considered a security vulnerability.

Examples of both categories will be looked at, though more of the first one will be reviewed than the second. We will try to look at the bug or, more generally, the cause and the resulting security vulnerability/flaw and possible effects.

3.1. Bug leading to a security vulnerability

This is the class we are addressing with this paper. Why would we want to take a particular look at this? Because these are the kinds of error that have made the news in recent years in different kinds of applications and have now become a real threat for security software. They usually open up the gates for malicious exploits and therefore it is crucial to check them out.

Buffer or heap overflows are the main part of this class, which may enable the execution of arbitrary and most likely malicious code or the performing a denial of service. Consequently, the examples presented here refer mainly to those bugs. We won't give a technical introduction to the terms used here, since we assume some general knowledge, and it isn't that important anyway, since we want to focus on the possible effects and not on the causes in this section.

ISS and the Witty worm [7]

Discovery date: March 8, 2004
Discovered by: *eEye Digital Security*

Several *ISS* products – firewalls and the like – contained a bug in the parsing of *ICQ* packets, which led to a stack-based overflow. It was possible to exploit this error and execute arbitrary code. This was then used by the Witty worm, which propagated via UDP packets using this vulnerability. The worm had a destructive payload which attempted to overwrite hard disk sectors. To sum it up, a security product opened up the gates for a worm.

The most interesting point about this incident, however, is that the vulnerability was found by a competitor of *ISS*, namely *eEye Digital Security*. According to them, it took *ISS* about 10 days to react and patch the vulnerability. The patch had been released one day before the worm was released. However, the worm still made a big impact and *ISS* received quite a lot of unwanted attention from the media.

Trend Micro VSAPI ARJ parsing [8]

Discovery date: February 23, 2005
Discovered by: *ISS*

An error in the ARJ file format parser when dealing with very long filenames led to a heap overflow, which in turn could be exploited to execute arbitrary code. This affected virtually every product that uses the VSAPI scan engine. An update of the VSAPI engine fixed the problem. A similar problem has been identified in *F-Secure* products [9].

McAfee virus library [10]

Discovery date: March 17, 2005
Discovered by: *ISS*

An error in the *McAfee* virus library when parsing LHA files led to a buffer overflow allowing the execution of arbitrary code. This could have been exploited by an email containing a crafted LHA file that targets this error. An update of the scan engine or recent virus definitions fixed the error.

Symantec multiple products UPX parsing engine heap overflow [11]

Discovery date: February 8, 2005
Discovered by: *ISS*

An error in the UPX parsing of the *Symantec AntiVirus* library could lead to a heap overflow, which can be used to execute arbitrary code by a remote attacker. Heuristic detection for a potential exploit using this flaw has been created and updates have been released.

Alwil Software Avast antivirus device driver memory overwrite vulnerability [12]

Discovery date: N/A
Discovered by: Piotr Bania

An error in the device driver could lead to a buffer overflow allowing the execution of arbitrary code or denial of service. This has been fixed by an update of the software.

Computer Associates Vet antivirus library remote heap overflow [13]

Discovery date: May 22, 2005
Discovered by: rem0te.com

An error in the *CA Vet* antivirus library could lead to a heap overflow when analysing OLE streams. This can be exploited to execute arbitrary code. The problem has been fixed with an updated engine version.

Kaspersky AntiVirus 'klif.sys' privilege escalation vulnerability [14]

Discovery date: June 7, 2005
Discovered by: *Softsphere*

It is possible to exploit a vulnerability under *Windows 2000* that allows execution of arbitrary code with high level privileges. An updated program version fixes the problem.

DataRescue Interactive Disassembler Pro buffer overflow vulnerability [15]

Discovery date: January 24, 2005
Discovered by: *iDefense*

When parsing the import directory of a PE file a buffer overflow could occur. This can be exploited to execute arbitrary code. An initial fix has been provided and a new version, released some time later, has fixed this error.

Compuware Softice (DbgMsg driver) local denial of service [16]

Discovery date: unknown
Discovered by: Piotr Bania

When working with debug messages an invalid pointer could be used to crash the system. A new version of the software fixed the problem.

Further examples include:

- *Symantec* security gateway DNS redirection [17].
- *OllyDbg* long process module debug vulnerability [18].
- *OllyDbg* 'INT3 AT' format string vulnerability [19].
- *PVDasm* long name debug vulnerability [20].
- *DataRescue IDA Pro* dynamic link library format string vulnerability [21].
- Multiple debuggers security bypass and code execution vulnerability [22].
- *Clam AntiVirus ClamAV* MS-Expand file handling DoS vulnerability [23].
- *Clam AntiVirus ClamAV* cabinet file handling DoS vulnerability [24].

These were just some of the examples of vulnerabilities found in recent months. Some things are apparent. Most of these vulnerabilities have been found by companies specializing in security analysis like *ISS* or *iDefense*. This leads to the question: why are those companies able to find these bugs while the security software vendors are not? Additionally, we have to assume that 'black hat' guys will use similar techniques and are probably finding and exploiting vulnerabilities as well. An interesting observation in this context is that many of these errors led to the execution of arbitrary code. Also, when an error is found, the chances are that there are similar errors in this software and even in the software of other vendors. This makes it easy for the 'black hat' guys to look for new holes to abuse. We hope security software vendors are following the released advisories and security alerts to prevent that.

3.2. What else? Or: not quite a bug, still a security vulnerability or indeed a bug, but not quite a security vulnerability.

This is the category that makes you think about the definition of security vulnerability presented in the first section. We will refer to misconfigurations or design errors that might lead to security vulnerabilities. We will also look at bugs that lead to unwanted behaviour, but which are not necessarily considered security flaws. And, finally, new options used by viruses need to be considered as well. These are neither bugs nor design errors or misconfigurations, just something no one came up with before. As you will see, there are quite a lot of very different examples for this category as well.

Trend Micro virus sig 594 causes systems to experience high CPU utilization [25, 26]

An error in *Trend Micro*'s Official Pattern Release (OPR) 2.594.00 led to slow downs or even 100% CPU utilization, therefore rendering the computer system inoperable when running a *Trend Micro* product.

How does this example fit in our paper? Users had two choices: turn off the *Trend Micro* product, which isn't exactly secure, or shut down the computer system and wait for an updated and fixed release, which would mean losses in business because of a security problem. While a fix was released shortly after, the problem still affected many, many computer systems and a lot of people didn't know what the problem was, leaving them clueless as to how to fix it. A few quotes illustrate the impact:

> 'This update took down virtually all 1,500 of our *Windows XP SP2* PCs and required many hours of work to resolve. The machines were rendered inoperable once this signature hit, and required many of us to work through Friday night.'
>
> 'How in the world could *Trend* [*Micro*] release a signature file that disables all *Windows XP SP2* machines? Why didn't [they] test this signature before it got released? I cannot believe that *Trend Micro* has no *XP SP2* machines to test on before they release patches, and if they don't they better get some ASAP. If this happens again, I can assure you that we will be finding a new antivirus vendor for our organization.' [27]

Windows NTFS alternate data streams [28, 29]

This is a completely different yet important example. A few months ago this became quite an issue and the public became aware of the usage of ADS by malicious hackers to hide their malware on infected computer systems. Some anti-virus solutions failed to scan ADS back then and didn't identify viruses hidden there. No one is to blame, as long as these are only theoretical threats (but ADS have been a known threat since 2000 [30]); however, when they are seen in the wild, vendors need to react.

Archive problems

There have been a number of different problems associated with the handling of archives in different products. *Symantec* had problems scanning manipulated RAR files, which led to a crash, therefore bypassing the file during a scan [31]. Other products skipped archives with invalid CRC checksums or had problems with unfiltered escape sequences in filenames contained in ZIP archives [32].

BitDefender bug bites GFI

An update in the *BitDefender* engine had the effect that GFI mail scanners deleted the body content of every incoming and outgoing mail, because the engine detected all mails as infected zip archives. As a consequence, *BitDefender* now plans to implement a testing module for integration [33]. That's something that might have been used before.

Further examples include:

- *Panda AntiVirus* deleting *Tobit David* communications software [34].
- *Symantec Brightmail AntiSpam* static database password [35].
- *McAfee Internet Security Suite 2005* insecure file permission [36].
- Easy DoS on *Kaspersky Anti-Hacker v1.0* [37].

These examples show some other problems of security software: architecture and design. Security isn't all about secure code. Security starts even earlier with the design of secure concepts and later implementing them. Also, the integration and composition of different components needs to be considered when aiming for security. Last, but not least, updates need to be reliable as well.

4. WHY ARE THERE BUGS AND SECURITY FLAWS IN SECURITY SOFTWARE, AND WHY SO MANY?

As mentioned in section 2, we have seen an increasing number of security vulnerabilities in security software in the past months, while the number in *Microsoft* products is decreasing. But security software is just following an industry-wide trend, which shows an increasing number of vulnerabilities (see Figure 1).

Some reasons for the increase in discovery of vulnerabilities in security software have already been mentioned and won't be presented again. This section will focus on the issue of why the bugs occur. This, of course, is not a problem that is unique to security software but rather an industry-wide one. Therefore we will first present some general reasons and then try to find out what makes security software especially prone to bugs.

There are several kinds of causes to be reviewed and [38] sums it up as three main factors:

- Technical factors – the underlying complexity of the task itself.
- Psychological factors – the 'mental models', for example, that make it hard for human beings to design and implement secure software.
- Real-world factors – economic and other social factors that work against security quality.

4.1. General reasons

This section will outline briefly some of the basic causes of errors and bugs, without going into technical details, partly following [38]. If you are familiar with software engineering or, for example, just followed the discussion about buffer overflows, you will know most of the stuff presented here since it should be common knowledge in that part of computer science. Nevertheless it needs to be presented to achieve a complete picture of the problems.

The simplest and most general reason is complexity. The bigger the software is, the more lines of source code are written and therefore the higher the chance of errors being introduced. It is, of course, never quite that easy, since the number of errors also depends on the kind and the language of code written, but it works as a rule of thumb. Furthermore, it's not just about implementation, it's also about the problem that perfect reliability and security is not achievable (yet?) and there is always a chance of errors, especially with more complex systems [38]. A nice quote that sums it up comes from *Microsoft* [39]:

> 'The core tenet of ASR (attack surface reduction) is that all code has a non-zero likelihood of containing one or more vulnerabilities. Some vulnerabilities will result in customer compromise. Therefore, the only way to ensure no customer compromises is to reduce code usage to zero. ASR is a compromise between perfect safety and unmitigated risk that minimizes code exposed to untrusted users. Code quality and attack surface reduction combined can help produce more secure software. Perfect code cannot do it alone.'

Another very common reason is the use of 'old code' that was written years ago and is still used in the software. For example, code to handle zip archives and the like (no wonder there are so many problems with that). But old code itself isn't the problem. The problem is that the old code is often unchecked and therefore not up to date with today's security requirements. When the code was written no one cared if a buffer overflow might be introduced, since the problem wasn't apparent yet. Some C functions are now considered insecure and should be replaced by their more secure equivalents. Unfortunately, these functions will introduce new problems, so they need to be handled with care as well.

The combination of separate components, which are secure by themselves, could introduce new security holes when put together. A famous example is the 'rlogin -l -froot' bug [38]. The 'BitDefender bites GFI' issue we presented above is another example.

It is also interesting to mention some of the psychological causes for errors in software. When looking for errors in code, you find those you are looking for, those you know about and those you know how to fix, but miss the others. This is the strength of automated security tests, since they could cover a broader range of possible error causes. Trust is another very important psychological issue – don't trust the user or the user input, until it has been verified by a trustworthy source. Finally, we tend to judge by our own personal experiences, but most likely future attacks aren't covered by our current experiences [38].

Last, but not least, there are real-world factors that need to be considered. One of the most important factors is tight deadlines. Even if you care about secure code and intensive testing of your software, there is still only limited time until the release of a new version or patch is scheduled. Sometimes it is necessary to settle with the best result possible and ship the software with possible errors [38]. A similar factor is the problem of being 'just secure enough', making the product just secure enough so people will still buy it. A great quote illustrates the point [38]:

> '"When are you guys going to stop shipping this crap?" he claims the answer he is proudest of was, "Sometime soon after you folks stop buying it."'

But it's also about features and competition. Time spent finding and fixing bugs is time lost for adding new functions. One can't really use security to advertise a product yet (although we hope this will change), but rather its numerous and cool features.

4.2. 'Vulnerability lifecycle' (security software specific)

With the general causes discussed we can now go on to the more specific ones. We've chosen 'Vulnerability lifecycle' as a caption for this section. We won't rely solely on this model but it will give us a good start. Let's have a look at Figure 2 below.

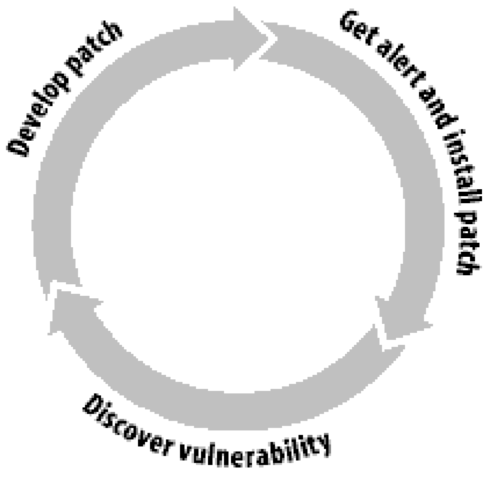

Figure 2 (source: [38]).

This outlines some of the basic problems vendors are facing today, and can be summed up as follows:

1. Someone uncovers and discloses a new vulnerability in a piece of software.
2. Bad guys quickly analyse the information and use the vulnerability to launch attacks against systems or networks.
3. Simultaneously, good guys start looking for a fix.
4. If the vulnerability is serious, or the attacks are dramatic, the various media make sure that the public knows that a new battle is underway.
5. Lots of folks get very worried. Pundits, cranks, finger-pointers, and copycats do their thing.
6. If a knee-jerk countermeasure is available and might do some good, we'll see a lot of it. (For example, CIOs may direct that all email coming into an enterprise be shut off.) More often than not, this type of countermeasure results in numerous and costly business interruptions.
7. When a patch is ready, technically-oriented folks who pay close attention to such matters obtain, test, and apply the patch. Everyday system administrators and ordinary business folks may get the word and follow through as well. Perhaps, for a lucky few, the patch will be installed as part of an automated update feature. But inevitably, many affected systems and networks will never be patched during the lifetime of the vulnerability, or will receive the patch only as part of a major version upgrade.
8. Security technicians, their attention focused, examine related utilities and code fragments (as well as the new patch itself!) for similar vulnerabilities. At this point, the cycle can repeat.
9. Weeks or months go by, and a piece of malicious software is released on the Internet. This software automates the exploitation of the vulnerability on unpatched systems, spreading without control across a large number of sites. Although many sites have patched their systems, many have not, and the resulting panic once again causes a great deal of business interruption across the Internet [38].

These are important points to consider. We start with a new vulnerability in software and soon there are attacks and exploits available. In the meantime, the developers will be working on a fix. In some very special cases, the media will take an interest in the vulnerability as well and publish information about it, hence providing publicity you weren't exactly hoping for. When a patch is released, most users will install it, but some systems might stay unpatched. Now people start looking for similar vulnerabilities, in the new patch too, and a new cycle will likely start. And every patch or new version introduces additional error sources and might lead to a vulnerability as well. '10% to 15% of all security patches themselves introduce security vulnerabilities' [38].

In general, this is something that applies to every vendor, not just the makers of security software. When there is a new version of a software program, many people don't update instantly and prefer to use the older version because they assume the new version may be buggy. They would rather wait until the new version has proven to be stable and any critical bugs have been fixed. However, this approach doesn't work for security software, since every update could be crucial for the company's security, leaving you with no choice but to install the update or shut down your computers to be 'secure'.

Let's take AV software as a prominent example. AV vendors need not only to release updates for their own bugs or vulnerabilities, they also need to release updates for new malware, and often several times a day. They also need to release program updates, in this case especially engine updates, much more often then most other vendors, since new kinds of attack are popping up constantly.

Just remember the jpeg issues which no one considered until it happened and required the programs to parse jpeg files. This has several consequences. First of all, with more patches released, the chances of errors being introduced is higher and more broken updates might be released. Also it is necessary to have every patch, signature or program update tested thoroughly. This is a lot of work, and 'normal' vendors already have problems keeping up with their own security patches every once in a while. It makes you wonder how security software vendors are actually dealing with this problem.

Just to compare numbers, some anti-virus vendors release 10–20 signature updates a day and several engine updates a month, while *Microsoft*, for example, releases usually only one to 10 patches per month. So this is quite an impressive achievement of the security software vendors. But of course we have seen enough examples that have shown us the limits of some vendors. Let's remember signature or engine updates that will render the system unusable, taking 99% of the CPU time or flagging certain clean files as false positives and deleting them in the worst case. This is certainly a testing issue that occurs because of the lack of time to do in-depth testing of every update. When a company schedules updates every few hours there is bound to be some trouble sooner or later.

While this is the most important issue, there are more that are unique to security software and need to be mentioned as well. Security software is dealing with malicious intents and inputs. This is something that should always be kept in mind. Security software shouldn't trust any input unless it can be verified. On the one hand, this refers to the malicious files that are scanned. See the aforementioned archive examples where some simple manipulations will let them slip through as clean files. But it also refers to the idea of the attack surface again. While other kinds of software just lock out potentially malicious inputs to reduce the attack surface, anti-virus scanners have to actively interact with malicious input to scan it and open up new possibilities for security flaws. And, finally, the application shouldn't trust the components of the software itself. Signature files can be manipulated unless they are cryptographically signed, but we don't see many of these yet. On the contrary, most of the vendors use very basic algorithms to cover the real content of their signature files, which don't pose a challenge to experienced virus writers. Also, integrity checks of the program files aren't carried out by all vendors. These and similar issues introduce additional security flaws which aren't really necessary.

Last, but not least, it seems that some vendors must get used to the fact that their software is now a target of security vulnerabilities. Granted, this is a relatively new threat to security software, but it needs to be handled with the same care and speed as, for example, outbreaks. It's just unacceptable when fixing a security-related bug takes several weeks or even more!

4.3. Conclusion

In addition to the usual causes of bugs and security flaws in software, there are a few specific causes in security software. The massive amount of updates is the most important, causing problems for both the vendor and the user. The vendor needs to make sure every update is ok and doesn't introduce new security issues, while the user doesn't, or just can't, trust newly-released updates and needs to carry out their own tests – or simply wait to update the systems. This leads to the concept of trust, which is one of the points that will be discussed in the next section, as well as some basic advice for both the vendor and the user.

5. WHAT TO DO?

Advising as to what to do is the tough part, since some of the advice given here may not be applicable and some is already in use. Further, most of the advice is theoretical and not applicable generally. But still it's necessary to give some direction on what is possible, for both the user and the vendor.

5.1. User

Let's have a look at the average home user of security software as well as the enterprise user. While they have some very common needs there are also some differences. The requirements they have in common are easy: they want the security software to secure them. To achieve this it is necessary to update the software constantly, as well as the signature files, in the case of an anti-virus software. This is where the differences occur.

While the home user may prefer an automated method of updating and can cope with a flawed update from time to time, the enterprise user has some different needs. The administrator in an enterprise environment cannot risk downtime of the computer system just because of a flawed update. So it is likely that their own tests will be carried out before an update is deployed. This, however, can take precious time and will increase the risk of infection when talking about viruses/worms.

On the other hand, there are still enough users that don't really care about updating their security software – they just aren't aware of the necessity. A nice report can be found in [40], which show how users actually update their systems. This PDF is especially interesting since it discusses a security hole of the *OpenSSL* toolkit which is usually used in security-aware environments. The outcome is somewhat surprising. While about 35% of the administrators updated their version relatively quickly (within one week, only 16% updated in the first two days), the other 65% percent didn't take any actions in the next 30 days [40]. The interesting thing however is, as soon as a worm that exploited the vulnerability (Slapper) was released, the updating and patching of the flawed versions started again and the percentage of vulnerable versions fell from 60% to 30%. The paper also discusses the causes for these events and comes to the same result as we did before [40]:

> '[...] it's sometimes undesirable to update immediately, since patches are flawed. Moreover, it may not be convenient to patch immediately.'

Both problems – flawed updates and users not deploying updates – lead to a simple yet effective piece of advice: use several layers of security. This starts at home with more than just one anti-virus solution and additional firewall software. At best a personal desktop firewall and a built-in firewall in your router or similar. But it is especially important in enterprise environments with different AV solutions on the gateways, servers and clients. The same goes for firewalls, again on the desktop, but also on the servers and gateways. Multi-scanner systems can help to improve security a lot here as well. This way it is possible to test updates first and make sure they don't have a negative impact on your system without being put at an instant risk if you don't install them at once.

Still, it would be better if the updates (or software in general) would always work (or at least not cause additional harm) and the user would/could trust them. These are the goals vendors should be aiming for.

5.2. Vendor

The two desired goals just presented will give the outline for this part. We will first take a look at the goal of working and secure (error-free or at least not introducing security vulnerabilities) software. Of course, this is not 100% achievable, but a few actions can be taken to get near it. In the second part we will take a short look at trustworthy computing and see how this is relevant for security software.

5.2.1. General actions

The first goal can be split into two parts. One concerns the generation of code, respectively the developing of software from design over implementation up to operation. The other part is about automation and testing. Of course, these are things that should usually be seen as a whole. But automation and testing become especially important because of the number of updates released in the security software sector.

Secure coding already starts with secure architecture and design. Of course there are many general and theoretical models known from software engineering regarding this and other problems. Luckily, some vendors have already implemented secure development methods. We will have a short look at the *Microsoft* way of doing it: The Trustworthy Computing Security Development Lifecycle [41], since it contains everything from architecture up to implementation, just what we need. *Microsoft* introduces four principles of secure development: secure by design, secure by default, secure in deployment and communications.

- Secure by design: the software should be architected, designed, and implemented so as to protect itself and the information it processes, and to resist attacks.
- Secure by default: in the real world, software will not achieve perfect security, so designers should assume that security flaws will be present. To minimize the harm that occurs when attackers target these remaining flaws, the software's default state should promote security.
- Secure in deployment: tools and guidance should accompany software to help end users and/or administrators use it securely. Additionally, updates should be easy to deploy.
- Communications: software developers should be prepared for the discovery of product vulnerabilities and should communicate openly and responsibly with end users and/or administrators to help them take protective action (such as patching or deploying workarounds) [41].

These are points we considered before but we have not yet shown how to accomplish them. Therefore, we will give a short overview of how *Microsoft* proposes it. The development lifecycle process is split up into different phases, starting with the requirements phase, over to design and implementation phases, followed by verification and release phase, and finally the support and servicing phase.

The requirements phase is, of course, the start of security work. Security milestones and criteria are decided, the integration of security and key security objectives are considered in this phase. Planning documents should be produced that define the security objectives explicitly.

The design phase will take the first ideas and requirements a step further. Security architecture and guidelines are decided as well as design techniques. In detail, these are things like 'layering, use of strongly typed language, application of least privilege, and minimization of attack surface', which are global things [41]. In addition, the security of individual elements should be specified and designed individually as well. Other steps of the design phase involve the documentation of the software's attack surface, threat modelling and defining additional ship criteria (regarding security).

The implementation phase is usually considered the most important, since the actual coding, testing and integration takes place here. Actions taken to remove security flaws at this stage are usually considered highly effective. Also, good design specifications start to pay off here, if proper threat modelling has been done this can be used to pay special attention to identified high-priority threats. Some of the detailed actions in this phase are:

- Apply coding and testing standards. Coding standards help developers to avoid introducing flaws that can lead to security vulnerabilities. Testing standards and best practices help to ensure that testing focuses on detecting potential security vulnerabilities rather than concentrating only on the correct operation of software functions and features.
- Apply security-testing tools including fuzzing tools. 'Fuzzing' supplies structured but invalid inputs to software application programming interfaces (APIs) and network interfaces so as to maximize the likelihood of detecting errors that may lead to software vulnerabilities.
- Apply static analysis code scanning tools. Tools can detect some kinds of coding flaws that result in vulnerabilities, including buffer overruns, integer overruns, and uninitialized variables.
- Conduct code reviews. Code reviews supplement automated tools and tests by applying the efforts of trained developers to examine source code and detect and remove potential security vulnerabilities [41].

The verification phase contains the start of user beta testing of the software. In the meantime, a 'security push' is started, as *Microsoft* calls it. This will include additional code reviews and security testing. Doing this in the verification phase ensures that all the testing and checking actually targets the final version of the version and some work in progress [41].

The release phase is the time of final security reviews, FSR, to make sure the software is ready to be delivered to the user. Possibly, code reviews and penetration testing should be conducted by outsiders to check for vulnerabilities. In the end, the FSR should give an overview of how well the software will withstand attacks. If vulnerabilities are found these should be fixed and the earlier phases should be reviewed again [41].

Finally, the support and service phase begins. The development doesn't end with the shipping of a product. Rather, the product team needs to react on discovered vulnerabilities in their software. This involves the ability to evaluate vulnerability reports as well as releasing security advisories and updates. On the other hand, it is also important to perform a post mortem analysis of the vulnerability and take appropriate action to prevent similar errors in the future. The main objective in this phase is: learn from your errors [41].

Something that hasn't been mentioned yet is education. While this should be common sense we still want to stress the fact. Educate yourself and your employees, follow security advisories and alerts and train your people. Black hat guys certainly will, so don't get left behind.

Lastly, we want to note some very short and specific actions as examples. Let's get back to the archive problems presented above. If scanning an archive may introduce buffer overflows, one might think about easy integrity checkers (not only for archive files, of course), as well as the possibility of easy filter rules. These rules might depend on the file's extension and the (basic) header, therefore not doing any actual parsing. Different handling methods could be configured, archive files could be let through on the gateways, in order to keep the gateways safe and not to risk any buffer overflows over there.

In the next step, the anti-virus software on the desktop could handle the file. Or possibly corrupted files can be put into quarantine and be checked with other tools later.

Another piece of advice for security-related software is the use of sandbox-like techniques as well as operating with minimal user rights. First, it should be impossible for any malware to 'evade' the virus scanner and cause harm just because it is being scanned. And second, if a malware evades nonetheless or causes a buffer overflow it shouldn't be able to do that with administrator rights.

Finally, yet another word on patches: many security products use third-party libraries like the *OpenSSL* toolkit which needs some security updates from time to time as well. Be sure to consider this when using third-party components and provide your users with a convenient way of patching, not just for your core product but for these components as well.

With these general actions presented we want to stress an important point for security software vendors. The testing of software, especially of updates, is one of the most important tasks for these vendors. This becomes obvious when we consider the number of updates released daily/weekly/monthly and compare it to other vendors. So invest a lot in your QA and testing if you want your users to trust your software – which in the end means they will buy your software. To make this point just a bit clearer we will take a very short look at trustworthy computing since this becomes important in a lot of ways. Just think about automatic updates, I know many security -aware people that disable those because they just don't trust them and don't want their system to break down because of a flawed update.

5.2.2. Trustworthy computing

In this part we will again refer to a *Microsoft* paper [6]. We will give a very short overview of why trust and trustworthiness is important for vendors and how this could possibly be achieved. All the details can be read in the paper mentioned above.

As we already mentioned, users need to trust software if they are supposed to use it. This is especially crucial when it comes to security-related software. If there is a security flaw in a media player, for example, the users might be ok with that, since security isn't the business of that vendor. But when there is a security flaw in security software, the user certainly won't be ok with that. When the security vendors can't even secure their own products, how are they supposed to secure the user?

No one wants additional risks because of flawed software. While normal software has the problem of trust only once in a while, with every new version released or when a security vulnerability is found, security software vendors face this problem maybe daily with every new update or patch they release. Each time the user has to decide once again if he trusts the vendor and whether he can install the update or not. And even if you gain the trust of the user, there is no guarantee that you will not lose it again.

So what can a vendor do to gain and keep the user's trust? *Microsoft* presents four goals from the user's point of view: 'security', 'privacy', 'reliability' and 'business integrity'. Followed by these goals are the means by which the industry should achieve these goals. Some of them we presented in the previous section, such as 'secure by design/default and in deployment', and also 'fair information principles', 'availability', 'manageability', 'accuracy', 'usability', 'responsiveness' and 'transparency'. Most of these are self-explanatory and otherwise they are discussed in detail in the *Microsoft* paper. In the last step, execution of these points is explained, which consists of 'intent', 'implementation' and 'evidence'. With this framework it should be possible to accomplish the challenges we presented.

6. CONCLUSION

What did we learn from this paper? Security vulnerabilities are an industry-wide problem, but they are increasingly affecting security software as well. *Microsoft* isn't the only target today, but rather any software that promises some success in exploiting. Additionally, next to these 'classic' threats of security vulnerabilities like buffer overflows, special problems of security software need to be considered. Every error could be security-relevant when it happens in security software! Also, the high number of updates introduces additional problems. This can lead to a loss in trustworthiness, which is an important resource for security-related software.

So, what should be done? First, all the classic measures should be taken to secure your code. Great publications have been released and industry-wide accepted standards have been developed to aid in that task. Secondly, the special problems of security software need to be addressed. This involves the update cycles that need very careful testing and quality assurance, as well as educating the users and winning and keeping their trust. The users need to use a responsible way of updating: 'Update often, update early, not too often and not too early though'. Last but not least, several layers of security should be used to cope with the failure of the one or the other layer.

Now, what to expect in the future? An increase in attacks and uncovered security vulnerabilities is a very likely scenario. The security holes uncovered by security analysts today might pose as examples for security holes uncovered by 'black hat' hackers tomorrow. This will eventually force vendors to act and react, and sooner or later this part of the game will end. But rest assured another game will already have begun ...

REFERENCES

[1] Burgess, Mark; 'Computer Security', Lecture Notes, 2001, http://www.iwar.org.uk/comsec/resources/security-lecture/show50b7.html.

[2] Wikipedia article 'Software security vulnerability', http://en.wikipedia.org/wiki/Security_vulnerability.

[3] Bishop, Matt; *Introduction to Computer Security*, Prentice Hall PTR, 2004.

[4] Jaquith, Andrew; 'Fear and Loathing in Las Vegas: The Hackers Turn Pro', 2005, http://www.yankeegroup.com/public/products/decision_note.jsp?ID=13157.

[5] Brenner, Bill; 'SANS: Security software increasingly vulnerable', 2005, http://searchwindowssecurity.techtarget.com/originalContent/0,289142,sid45_gci1084940,00.html?bucket=NEWS.

[6] Mundie, Craig; 'Trustworthy Computing - Microsoft Whitepaper', 2002, http://www.microsoft.com/mscorp/twc/twc_whitepaper.mspx.

[7] http://xforce.iss.net/xforce/alerts/id/167.

[8] http://www.trendmicro.com/vinfo/secadvisories/default6.asp?VName=Vulnerability+in+VSAPI+ARJ+parsing+could+allow+Remote+Code+execution.

[9] http://xforce.iss.net/xforce/alerts/id/188.

[10] http://xforce.iss.net/xforce/alerts/id/190.

[11] http://xforce.iss.net/xforce/alerts/id/187.

[12] http://pb.specialised.info/all/adv/avast-adv.txt.

[13] http://www.rem0te.com/public/images/vet.pdf.

[14] http://www.softsphere.com/security/.

[15] http://www.idefense.com/application/poi/display?id=189&type=vulnerabilities&flashstatus=true.

[16] http://pb.specialised.info/all/adv/sice-adv.txt.

[17] http://securityresponse.symantec.com/avcenter/security/Content/2005.03.15.html.

[18] http://www.securityfocus.com/archive/1/393747.

[19] http://pb.specialised.info/all/adv/olly-int3-adv.txt.

[20] http://neosecurityteam.net/Advisories/Advisory-10.txt.

[21] http://pb.specialised.info/all/adv/ida-debugger-adv.txt.

[22] http://www.security-assessment.com/Whitepapers/PreDebug.pdf.

[23] http://www.idefense.com/application/poi/display?id=276&type=vulnerabilities.

[24] http://www.idefense.com/application/poi/display?id=275&type=vulnerabilities.

[25] http://www.trendmicro.com/en/support/pattern594/overview.htm.

[26] http://isc.sans.org/diary.php?date=2005-04-23.

[27] http://www.zdnet.co.uk/print/?TYPE=story&AT=39196220-39020375t-10000025c.

[28] http://www.securityfocus.com/infocus/1822.

[29] http://www.heise.de/security/artikel/52139/2.

[30] http://securityresponse.symantec.com/avcenter/venc/data/w2k.stream.html.

[31] http://securityresponse.symantec.com/avcenter/security/Content/2005.04.27.html.

[32] ftp://ftp.aerasec.de/pub/advisories/unfiltered-escape-sequences/.

[33] http://www.theregister.co.uk/2005/03/02/gfi_beserker/.

[34] http://www.computerpartner.de/news/226206/index.html.

[35] http://securityresponse.symantec.com/avcenter/security/Content/2005.05.31a.html.

[36] http://secunia.com/advisories/14989/.

[37] http://cert.uni-stuttgart.de/archive/bugtraq/2003/03/msg00264.html.

[38] Graff, Mark G.; and van Wyk, Kenneth R.; *Secure Coding: Principles & Practices*, O'Reilly, 2003.

[39] Howard, Michael; 'Attack Surface - Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users', *MSDN Magazine*, November 2004, http://msdn.microsoft.com/msdnmag/issues/04/11/AttackSurface/default.aspx.

[40] Rescorla, Eric; 'Security holes... Who cares?', *Proceedings of the 12th USENIX Security Conference*, 2003, http://www.rtfm.com/upgrade.pdf.

[41] Lipner, Steve and Howard, Michael; 'The Trustworthy Computing Security Development Lifecycle', 2005, http://msdn.microsoft.com/security/default.aspx?pull=/library/en-us/dnsecure/html/sdl.asp.