vb 2019 LONDON 2 – 4 October 2019

# OOPS! IT HAPPENED AGAIN!

*Righard Zwienenberg*
ESET, The Netherlands

*Eddy Willems*
G DATA Software, Belgium

righard.zwienenberg@eset.com; eddy.willems@gdata.de

## ABSTRACT

The problems malware causes for the digital ecosystem began three decades ago. Although some viruses are known to be older than that, they didn't really cause too much trouble. In the last three decades we have had many (new versions of) operating systems as well as new versions of applications. Usually one learns from one's mistakes, but somehow in the cyber-ecosystem the opposite seems to be true.

In this paper we will take you on a scenic tour spanning three decades of malware, three decades of digital ecosystems and three decades of history repeating itself. When will we ever learn?

## INTRODUCTION

Malware has been causing problems for three decades, yet we still keep making the same mistakes. You say we don't? Hmm… Let's start by taking a look at something large from 1997.

## MACRO VIRUSES

In 1997 the world first experienced macro viruses for *Microsoft Office* with the (in)famous W97M/Melissa.A, written by David L. Smith [1]. It was not the most complex virus in the world, 'just' mass-mailing itself to the first 50 contacts in your *Microsoft Outlook* address book, but it happened. There is a first time for everything, and this was surely unprecedented. The entire world experienced it – you couldn't even switch on the television news without hearing about it. You would think that, from this episode, we would have learned not to assume that any email that originates from a friend or business acquaintance is safe.

Three years later, in 2000, VBS/Loveletter.A [2] did the same thing, except this time it spread to all the contacts in your address book. Some social engineering and a new file type (VBS: Visual Basic Script) seemed to be enough for everyone to forget the lessons learned from the Melissa outbreak. Mankind seems to learn slowly – but we do learn, don't we?

The following year, VBS/AnnaKournikova [3] was released by the Dutch teenager Jan de Wit. Although it was built using a virus creation kit, and just by pushing some buttons therein, the perfect piece of social engineering made everybody forget any lessons that had been learned from previous outbreaks, once again. The email attachment purported to show nude pictures of the then famous

tennis star Anna Kournikova. Of course, that wasn't the case – double clicking the attachment simply made the worm spread further to all your contacts.

These examples demonstrate that most people don't learn from history and that the right story (social engineering) causes them to repeat the same mistakes again. And when we look at current threats, we see that macro malware is back once again. Just over two decades after it wreaked its initial havoc, people are falling for it again and again. Most of the time this happens due to people implicitly trusting communications that appear to come from friends or acquaintances, but it also happens because the operating system or application in question contains bugs and needs a patch or an update.

## BUGS, EXPLOITS AND PATCHES OR UPDATES

If you pay attention to IT news, you will know that not a week goes by without a big story being reported about newly discovered vulnerabilities in commonly used software. Everyone (the expert and layperson alike) fears these so-called *zero-days*. The term refers to the time that elapses between the discovery of a leak or vulnerability and the availability of a patch that fixes the problem. Such issues are feared because it is assumed that cybercriminals are more likely to write malware that exploits a leak before the software vendor can come up with a patch. It is, of course, a realistic scenario that should indeed make you nervous. But zero-days are far from being the biggest problem.

For cybercriminals, information about security vulnerabilities in computer programs is highly sought after. If they succeed in spreading malware that exploits a specific flaw in system defences before the affected software vendor can counter this with an update, the chances of a successful attack or a malware pandemic are much greater. However, such malware may be caught by anti-malware programs before the affected vendor is able to issue a patch. In fact, anti-malware products may be able to detect unknown malware using the vulnerability before a patch is released, since it's sometimes easier and quicker to detect the presence of an exploit than it is to generate an effective and thoroughly tested patch for the affected software. This is, of course, by no means a criticism of affected vendors, it's just how things are in the world of programming. Some coding and QA tasks are more time-consuming than others. And it's still essential that you install a patch when it becomes available, rather than rely on detection of malware (it's rather like the difference between applying an emergency field splint to a broken limb and resetting the bone under hospital conditions).

But here's the thing: research has shown that a high percentage of successful attacks on business networks have made use of security breaches for which, at the time of the attack, there is *already* a perfectly adequate mitigation for the vulnerability in question (a so-called *patch*, the software equivalent of the patch used to seal a leak in an inner tube) available from the manufacturer of the software. The big security problem is therefore not that there are no solutions available, but that patches are not installed as soon as is practicable after they become available. Ironically, cybercriminals take advantage of patches: by analysing the patches made available by the software producers, they get exactly the knowledge they need to make malware that exploits those vulnerabilities. Then they unleash their malware into the world in the hope (and certain knowledge) that many companies will not have rolled out the patches yet.

The fact that patches are not always installed properly or promptly on corporate networks is understandable. IT administrators often have no idea which software is used by all the company's employees, and they are even less likely to have a good overview of exactly which software versions are running on which machines. Moreover, patches can conflict with certain (less common) programs

that are used by companies, or a patch may disable (intentionally or as a side-effect) certain functionalities that are important to some users, so it may be easier just to skip the patch. After all, as we often say in IT circles: 'if it ain't broke, don't fix it'! But the patches discussed here are intended to fix something that *is* broken, and all companies should realize how important it is to roll out security patches or updates as soon as possible.

## DO YOU REALLY LIKE UPDATES?

We are living in a world of updates and these days it seems we need to update or patch several pieces of software nearly every day. If it's not on our *Windows* or *Mac* systems, our *Android* or *iOS* mobile devices or even our smart TVs or smart speakers will ask us to update very frequently, and it may be almost every day depending on what apps you have installed on your device. But what is the real reason that we need to update or patch so much?

Possibly the simplest and easiest explanation is that software is created by humans, and humans make mistakes. This has been the case since the beginning of computing.

Many will already know the story of where the word 'bug' came from in computing circles. There were already buggy programs in the days of the very first computers in the 1940s. Computers then filled entire rooms and worked with electromechanical relays and countless vacuum tubes. The IT teams at the time included an electrician whose full-time job was replacing vacuum tubes. The choice among programmers at that time to use the word 'bug' for a disturbing error is generally attributed to Grace Hopper of the University of Harvard, one of the very first programmers. In her Harvard Mark II logbook entry for 9 September 1947, the electrician stuck a moth that had caused a short circuit. It was the first 'bug' in the sense of computer error.

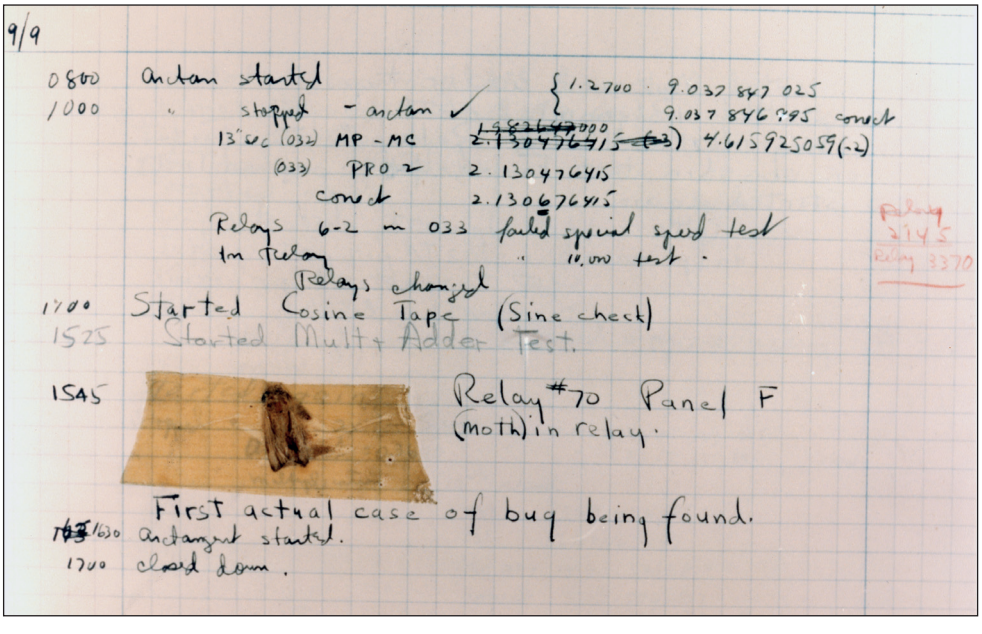


*Figure 1: Grace Hopper's logbook showing the first computer 'bug' [4].*

## THIS IS THE FIRST BUG!

But Hopper's bug appears not to have been the very first computer failure, according to the *Two-Bit History* blog [5]. The very first programmer, Lady Ada Lovelace, born Byron, had already written it in 1843, it now appears.

Ada Lovelace was the first, and only legal, daughter of the poet Lord Byron, but shortly after her birth her parents separated, and it was her mother who encouraged Ada's interest in mathematics and science. At the age of 17 Ada met Charles Babbage, the designer of the first mechanical calculators that, controlled by punch cards, were able to perform a number of steps one after the other and use the results from one step for the next. However, his designs required more precise tolerances than the blacksmiths and metal founders of the time could supply, and his '*analytical engine*' was never built. His work was forgotten and had no influence on the design of the first computers in the 1930s-1940s. Only later was it rediscovered, and in 1991 the London Science Museum rebuilt one of his designs. It worked perfectly.

### Loop

After meeting Babbage, Ada Lovelace had begun to think about how to operate such an analytical engine, with variables that could get different values – values that had to be kept up to date – and with groups of calculation steps that could be repeated. In other words, she invented the 'loop', today an important basic part of every program. She also immediately came up with the 'loop within a loop'. She also realized that with such a machine you could do more than count – making music, for example. But she had to write her programs in a vacuum – for a device for which she had a description but which did not exist in practice. Programmer Sinclair Target translated her complex program for calculating Bernouilli numbers into the modern computer language *C*. Only, it didn't run for a second.

The mistake turned out to be with Lovelace. In the fourth step of her program she says that $V_5$ must be divided by $V_4$, while in fact it must be the other way around. A minor mistake for a programmer, but the first bug for humanity.

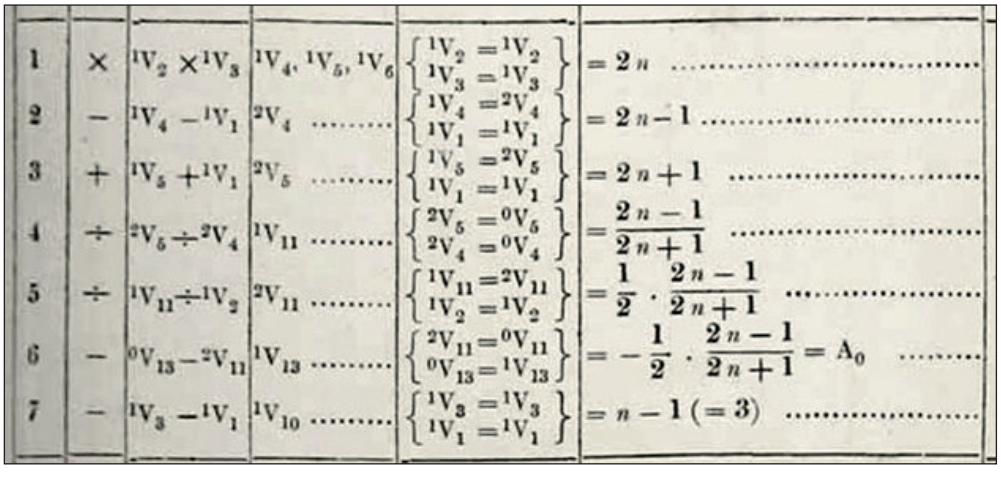

*Figure 2: Ada Lovelace's program [6].*

## THE QUESTION REMAINS: DID WE LEARN ANYTHING, OR WHEN WILL WE LEARN?

We are humans and we make mistakes. For programmers this sometimes results in bugs in software that have the potential to be misused or exploited by cybercriminals and hackers. Even with very good update mechanisms in place we are still solving problems which wouldn't have arisen had security by design been used from the beginning. Still, we have our doubts as to whether humanity will ever achieve bug-free and error-free software creation and handling. This is why updating (or patching) will remain very important in the software and online world. We should all try to be better focused to optimize and protect the update processes of all our software in the future.

## #FAKE

In the last five years there has been an uptick of fake news. Due to social media on the Internet, news, whether fake or true, spreads rapidly all over the globe. Fake news can be considered successful, harmless, and even funny when it is used, for example for April Fool's Day, but at other times it can cause serious damage – damage to reputations, damage to sales of companies, even damage to fair elections. If we believe certain countries, other nation states have spread false information in order to influence the decision of the voter. But of course, that is nothing new either, although previously we knew it as propaganda – false or misleading information has historically been used to deliberately misinform people. For example, in Germany old footage was shown in cinemas during World War II to support the Nazi Army's claiming of victories when in fact they were losing the war, just to keep up the morale of its soldiers.

It seems that fake news has always existed, and yet we have still not got used to it. In the cyber era, the phenomenon has only become more widespread and taken on different forms. Recently, the well-known online marketplace *Amazon* has been 'suffering' from a flood of fake five-star reviews. Consumers trusting the reviews listed with the products will be misled. An experienced *Amazon* user may be able to spot that the (often) thousands of reviews are all unverified and posted within the last few days, but in reality the high number of five stars and the low(er) price for the item encourages the user to press that 'Buy Now' button. A seller can buy fake reviews cheaply and easily on the Internet (Fake as a Service); the victims are *Amazon*, the buyer, and the producer of the genuine (safe) item.

It is amazing how easy it is to obtain and deploy fake reviews. We should have learned from the past and prevented it. A flood of fake reviews should not be a problem in the era of big data, artificial intelligence and machine learning. A very simple rule is that only people that have bought an item can leave a review. Another rule is that when there is a flood of reviews within a few hours or days they should be treated as suspicious… you get the picture.

## CURRENT-DAY THREAT SCENARIOS

These days, no threat scenarios are new. All of them have happened before. They may have happened within a different or previous operating system, previous or different versions of applications, etc., but there is nothing entirely new. Except they seem to have moved more to the supply chain. The threats here do not just come from cybercriminals, but even from the genuine software industry where the most famous incident by now is the Telebots backdoor that caused a worldwide ransomware outbreak after the infrastructure of a legitimate Ukrainian accounting software firm was compromised [7].

Such supply-chain threats have happened to operating systems, too. Spectre/Meltdown [8, 9] required microcode updates not just for *Intel* processor architectures [10], but for other processor architectures as well. Updates like these require rigorous testing as faulty microcode can crash a system, and this is exactly what happened in some circumstances [11].

You might expect that events like this would increase awareness and enhance the testing of updates, but during 2018 and 2019, we have seen a series of operating system updates that had to be revoked as they caused a lot of problems.

## CONCLUSION

It is obvious, in the cyber-ecosystem, that there is a lot of 'evolution'. Over the past three decades (and longer) we have seen new versions of operating systems and bugs but also malware and cyberattacks. Still the basic problem seems to be related to humans. New cyber-problems are not always as new as they seem to be. Humans forget, most of us don't think back more than ten years. Most of us forget that we saw some relatively large cyberattacks 20 or even 30 years ago. Humans make mistakes and we will continue to do so, which leads to insecurity in the always evolving world. (Cyber)criminals will always be ready to explore new technology and new techniques, but the basics remain the same. Burglars will be burglars even if they use the newest, most advanced techniques. And there is indeed evolution on either side: the use of AI (Artificial Intelligence) on both the good and the bad side is an example of this. Human awareness of the always-evolving, new techniques and the recurring use of old techniques is constantly needed. **Continuous awareness education** is key in this and could at least change the way we think. A cyber-problem always[1] [12] has a human factor and a technological factor, but it's clear by now that the two factors will always be connected to each other and result in further cyber-problems, even in the future.

The latest trend is **Corporate Social Responsibility (CSR)**, where the major vendors are giving back to the customers, most often in the form of education and awareness programs. A movement we highly encourage because, as may be obvious from this paper, 'things' happen again, perhaps a decade or more later, and with proper awareness and education, renewed attack attempts may become futile.

## REFERENCES

[1] Melissa (computer virus). Wikipedia. https://en.wikipedia.org/w/index.php?title=Melissa_(computer_virus)&oldid=896844874.

[2] ILOVEYOU. Wikipedia. https://en.wikipedia.org/w/index.php?title=ILOVEYOU&oldid=898545899.

[3] Anna Kournikova (computer virus). Wikipedia. https://en.wikipedia.org/w/index.php?title=Anna_Kournikova_(computer_virus)&oldid=825224949.

[4] NH 96566-KN The First "Computer Bug". Naval History and Heritage Command. https://www.history.navy.mil/our-collections/photography/numerical-list-of-images/nhhc-series/nh-series/NH-96000/NH-96566-KN.html.

[5] What Did Ada Lovelace's Program Actually Do? Two-Bit History. https://twobithistory.org/2018/08/18/ada-lovelace-note-g.html.

[1] Eddy Willems' Second Law mentioned in his book 'Cyberdanger [12].

[6] Ada Lovelace. Wikipedia. https://en.wikipedia.org/w/index.php?title=Ada_Lovelace&oldid=897378212.

[7] Cherepanov, A. Analysis of TeleBots' cunning backdoor. We Live Security. https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/.

[8] Spectre (security vulnerability). Wikipedia. https://en.wikipedia.org/w/index.php?title=Spectre_(security_vulnerability)&oldid=897132195.

[9] Meltdown (security vulnerability). Wikipedia. https://en.wikipedia.org/w/index.php?title=Meltdown_(security_vulnerability)&oldid=897142316.

[10] KB4100347: Intel microcode updates. Microsoft Support. https://support.microsoft.com/en-us/help/4100347/intel-microcode-updates-for-windows-10-version-1803-and-windows-server.

[11] KB4100347 causes infinate autorepair loop on XeonE5-2690 on x79 booting from NVMe. https://answers.microsoft.com/en-us/windows/forum/windows_10-update/kb4100347-caueses-infinate-autorepair-loop-on/2ae27ac5-12fc-43b6-a6bc-d3ae0774d724.

[12] Willems, E. Cyberdanger. Springer 2019 (hardcover: ISBN 978-3-030-04530-2, eBook: ISBN 978-3-030-04531-9).