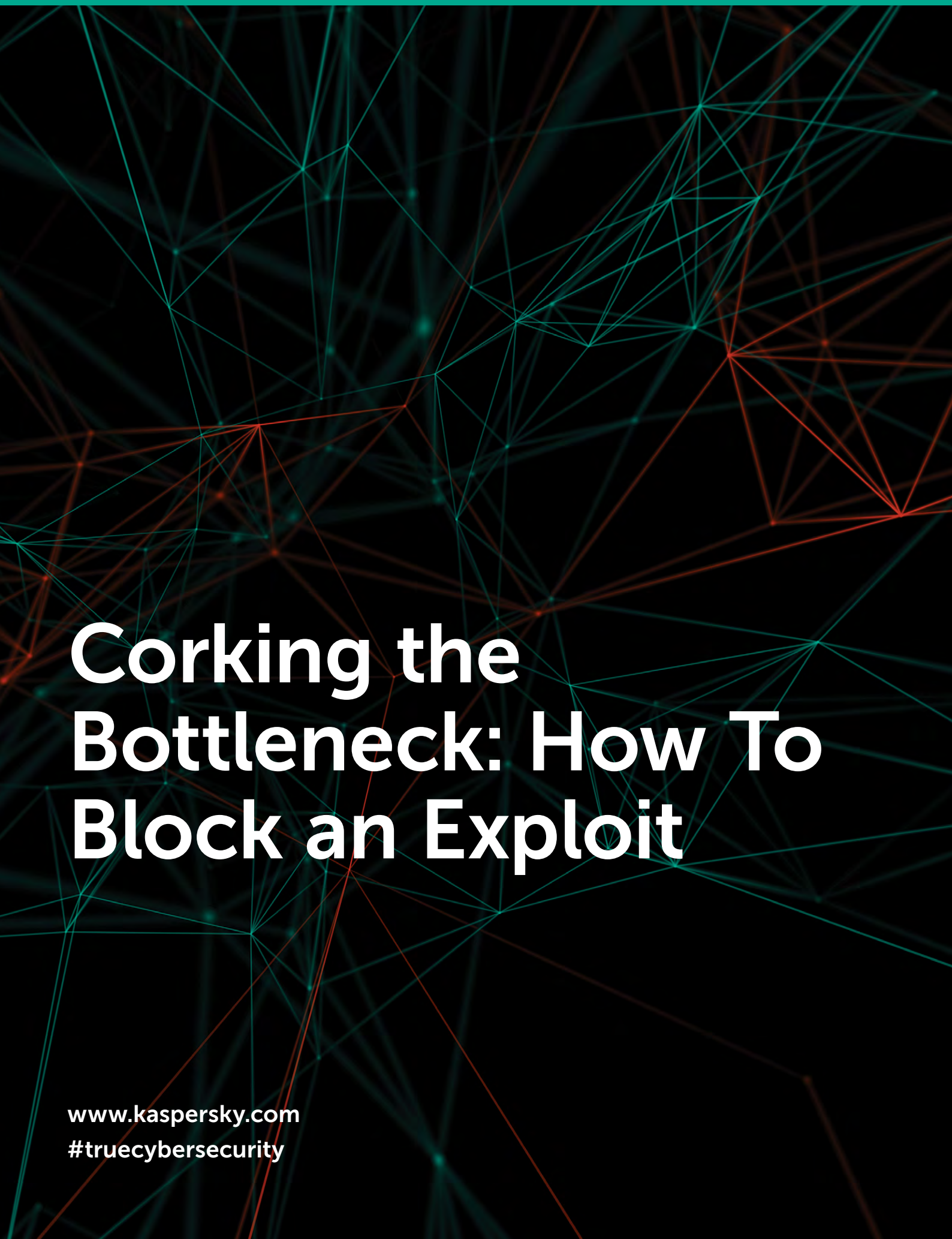
KASPERSKY lab



# Corking the Bottleneck: How To Block an Exploit

www.kaspersky.com
#truecybersecurity

# Corking the Bottleneck: How To Block an Exploit

## Exploits as a Part of the Modern Threat Landscape

Despite some more recent trends, such as the heavy use of non-malware components in their toolsets, cyberattackers as a whole retain their reliance on vulnerability exploitation techniques as an attack spearhead, the primary means for initial penetration. In 2016, the use of exploits increased overall by roughly a quarter, and for corporate users alone the percentage was even higher.

The most widespread scenarios remain the same: either exploit-carrying attachments to e-messages, or drive-by attacks, including malicious links and 'watering hole' redirects to the same links, with exploits on hand to attack whatever vulnerabilities the victim's system possesses. And it's long been obvious that, while attackers may get lucky and find a zero-day, most of the time it's known vulnerabilities that are exploited. The fact is that the probability of finding an unpatched OS or old app still in use is high enough to make these attacks well worthwhile. For example, a well-known CVE-2010-2568 vulnerability once used by Stuxnet, remains the top performer in terms of number of users attacked, and the recent WannaCry ransomware pandemic also used a vulnerability that has an available patch. Then again, even the most security-aware companies sometimes have vulnerable software running in business-critical processes – for a number of reasons, including complicated update procedures, compatibility issues or configuration-sensitive legacy applications.

So protection against vulnerability exploitation remains a primary concern for Endpoint security. After all, the whole data breach prevention task turns on how effectively the attack's spearhead can be blocked.

## The exploit's own kill chain and counteractions at each phase

Of course, for the end user, the best possible outcome is all that really matters: if the existing security solution successfully prevents the attacker from doing anything malicious, a victory is scored.

But for the solution's vendor, this particular phase of a cyberattack is a very delicate moment - it involves the user's own applications, which all too often don't tolerate unsubtle handling and can react with crashes and The Blue Screen of Death. Every phase of an exploit's own 'kill chain' presents different opportunities and challenges for the defender. Let's look at this in more detail:

### Delivery

Typically an email attachment or website, as above. This ends with the targeted application beginning to process the offered data – including the exploit code.

**Counteraction**

Some exploits can be blocked during this phase using proper mail server security, anti-phishing and content analysis. A considerable amount of mass malware is, in fact, blocked here. But more sophisticated specimens, especially in well-prepared targeted attacks, can shake static analysis mechanisms off their trail. The dynamic study of everything that comes your way in a sandbox is a good move, but, to be really effective, it requires considerable resources and skill. Also, in most scenarios, it would not allow *blocking* but would only *alert* corporate security officers, which, given the time gap between delivery and actual sandbox detonation (there's usually a queue), means this is not a true *defense* against exploits.

### Memory manipulation

During this phase, rogue data is placed into different memory areas. This is not a violation of any security principles and is mostly harmless in itself – but at a later stage, once the vulnerability has been exploited, this data is used in specific attack processes.

**Counteraction**

There are only a limited number of ways to insert this rogue data, and all are well known, so modern Operating Systems offer built-in mitigations to counter exploits at this phase. But, to be effective, these mitigations require applications to be compiled with particular parameters in a modern development environment. Unfortunately, this is not possible for some older apps. Some mitigations can be invoked externally, but the flip side of the coin is that externally forcing memory use restrictions can result in instability and crashing the application that the security solution is trying to protect.

### Exploitation

Here's where activities which aren't part of the general order of things start happening. Using the existing vulnerability, the attacker coerces the attacked app's process into performing actions, which may or may not be within its standard capabilities, but, in any case, are instrumental in the attacker's progress. Depending on the attack scheme, this is usually followed by shellcode execution, though in some cases standard app functionality can be leveraged to deliver the malware payload from the C&C server.

### Counteraction

To be able to detect and influence activities occurring during this phase, the security solution must have access to the protected process's context. The only way to do this efficiently is by performing a process injection, not unlike the malware technique itself. While this approach offers an opportunity to stop exploits at an earlier stage and allows the protection of arbitrary processes, it also suffers from considerable drawbacks. Performance degradation and compatibility issues are not infrequent, and their probability increases with each mitigation technique switched on for a particular process. Also, for processes the solution was not previously tested with, the need for tedious trial-and-error configuration can cause great inconvenience, especially for hard-pressed generalist IT administrators. Some vendors strongly recommend consulting with their support teams prior to any attempts at mitigation rule customization.

It is also worth noting that every new version-change of the application can result in unexpected crashes and the need to either tweak security settings to find a safe configuration – or to refrain from using this mechanism until (if ever) the solution's vendor manages to adjust it sufficiently.

## Shellcode execution

This is where the attacker's arbitrary code is executed, resulting in the execution of a malicious payload.

### Counteraction

This is where the exploited process starts doing things it's mostly not expected to, and this can be detected externally, using non-invasive behavior tracking mechanisms. Such mechanisms usually don't require any manual configuration, which saves a lot of time and effort for IT staff. Also, there's no meddling with the protected process itself, so there's zero chance for compatibility and performance issues. In fairness, it should be noted that, besides understanding what activities should be treated as suspicious, this approach's effectiveness also depends on a knowledge of what the process *normally does*, so it's hardly suitable with previously unknown apps. But then again, 99.9% of exploitation scenarios target quite a limited number of popular applications and system components, so the gains of hassle-free defense clearly outweigh the limitations. This approach can also benefit from additional sources of threat intelligence, such as lists of known attack-related hosts and IP addresses.

## Payload execution

The payload can be downloaded as a file – or, in the case of a fully bodiless scenario, it can be loaded and executed directly in the system's memory. After this point, the malicious activity starts.

### Counteraction

Launching another application or execution thread can look suspicious, especially if the app in question is known to lack this functionality. So this may well serve as a pretext for a security solution to set execution to 'pause' and to analyze the legality of the launch in more detail. Additional behavioral indicators provided by execution tracking mechanisms should allow the solution to block the payload execution with confidence.

This particular phase applies to the whole plethora of exploitation scenarios – any exploit's ultimate goal is to launch a payload. So this becomes a bottleneck - leaving the attacker with very little space to maneuver. Despite the fact that the exploitation has already happened, the whole attack sequence is at its most vulnerable right now.

As you can see, different phases of the exploit's micro-kill chain may require different counteraction mechanisms. While we, in Kaspersky Lab, consider a multi-layered approach to cybersecurity the most effective, we also understand that providing the best outcome for customers means not only reliable protection, but also the lowest possible impact on existing business processes.

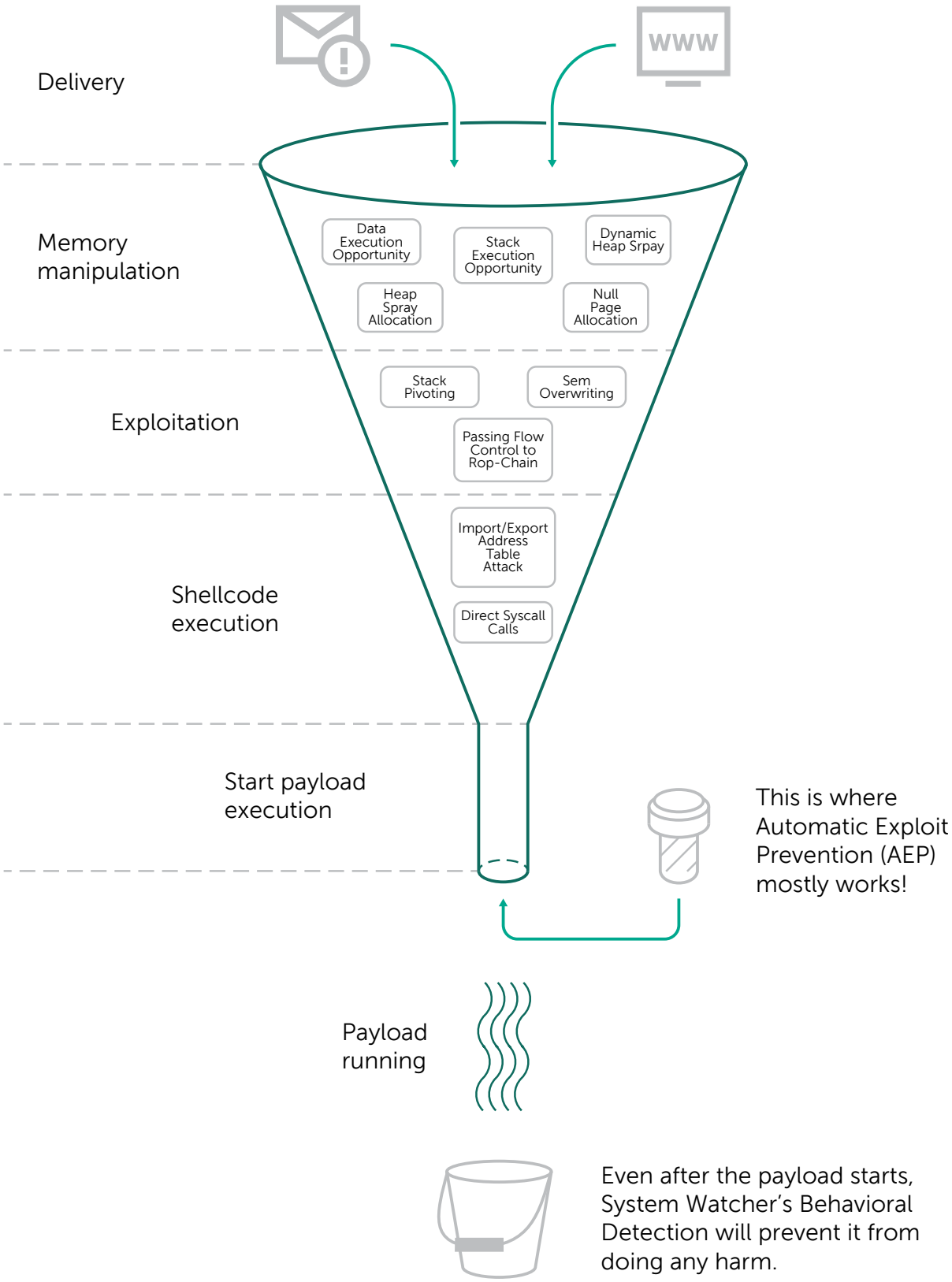
And so we created our Automatic Exploit Prevention (AEP)[1], a multi-layered system in its own right that utilizes not only the most effective, but also the most reliable, resource-efficient and hassle-free combination of techniques to ensure a smooth experience for both end users and administrators.[2]

# Attack techniques, zero-days and Kaspersky Automatic Exploit Prevention (AEP)

While it intuitively feels safer to block an exploit as early as possible in its kill chain, the techniques for doing this actually pay off much less frequently than we'd wish. Multiple compatibility issues and issues around any change in the protected apps led us at Kaspersky Lab to decide on avoiding most of these mitigations and focusing on non-invasive behavioral prevention. It's also important to remember that these mitigations work with previously known technique classes, which are common knowledge. So when it comes to a zero-day exploit, using something out of the ordinary, they're likely to be sidestepped by the attacker. Many of these mitigation techniques used by vendors other than Kaspersky Lab are, in fact, very similar to those used by well-known Microsoft EMET – and there are multiple PoCs showcasing exactly how they can be outmaneuvered.

[1] Automatic Exploit Prevention is available in Kaspersky Endpoint Security for Business and Kaspersky Security for Virtualization Light Agent

[2] Here you can find the result of independent testing by MRG Effitas and AV Comparatives.


Delivery
WWW
Memory manipulation
Data Execution Opportunity
Stack Execution Opportunity
Dynamic Heap Srpay
Heap Spray Allocation
Null Page Allocation
Exploitation
Stack Pivoting
Sem Overwriting
Passing Flow Control to Rop-Chain
Shellcode execution
Import/Export Address Table Attack
Direct Syscall Calls
Start payload execution
This is where Automatic Exploit Prevention (AEP) mostly works!
Payload running
Even after the payload starts, System Watcher's Behavioral Detection will prevent it from doing any harm.

On the other hand, behavioral detection uses a number of indirect indicators. Kaspersky Automatic Exploit Prevention, for example, can also leverage additional sources of information provided by different security layers, such as tracking changes in particular memory areas, addressing suspicious URLs and so on. Both independent testing and multiple real world cases prove that Kaspersky Automatic Exploit Prevention successfully detects both synthetic and real zero-day exploits, despite having no previous knowledge of the attack. Also, thanks to Automatic Exploit Prevention, Kaspersky products demonstrate outstanding effectiveness against exploit-using ransomware families, such as CryptXXX, during the earlier campaign stages when no information about these attacks has had time to build up.

Still, some mitigations don't require process tinkering and the heavy use of resources, and are safe to use with certain applications – so AEP makes use of these as well.
Here's how Kaspersky Endpoint Security armed with AEP withstands different attack techniques (note that not all of them are, in fact, directly related to vulnerability exploitation):

| Mitigation\| attack technique | Kaspersky Endpoint Security – mitigation | Kaspersky Endpoint Security – prevention | Result |
|---|---|---|---|
| DEP (Data Execution Prevention) \| Buffer overflow | PROVIDED BY OS (majority of the modern applications have DEP enabled by default. Turning DEP on for applications for which it was not designed may render it inoperable) | | **Protected** |
| ASLR (Address Space Layout Randomization) \| Leveraging data at predictable locations | YES | | **Protected** |
| Stack Pivot \| Stops abuse of the stack pointer | YES | | **Protected** |
| Null Page Allocation \| Null page exploit | PROVIDED BY OS (Unsafe to mitigate externally) | | **Protected** |
| Heap Spray Allocation \| Placing shellcode copies at as many memory locations as possible | | YES | **Protected** |
| Dynamic Heap Spray \| Stops attacks that spray suspicious sequences on the heap | | YES | **Protected** |
| SEHOP (Structured Exception Handler Overwrite Protection) \| Structured Exception Handler (SEH) overwrite | PROVIDED BY OS (SEHOP is provided by Windows OS started from Windows Vista SP1 (all modern office productivity applications and browsers are protected by SEHOP by default) | | **Protected** |
| ROP \| Stops Return-Oriented Programming attacks | | YES | **Protected** |
| Load Library \| Prevents loading of libraries from UNC paths | PROVIDED BY OS (due to compatibility issues, mitigations are provided by OS – https://support.microsoft.com/en-us/kb/2264107) | | **Protected** |
| DLL Hijacking | | YES | **Protected** |
| Reflective DLL injection | YES (detected by System Watcher) | | **Protected** |

| Mitigation\| attack technique | Kaspersky Endpoint Security – mitigation | Kaspersky Endpoint Security – prevention | Result |
|---|---|---|---|
| Network DLL \| Loading malicious libraries by placing them on network paths | | YES (Legitimate functionality abuse, detected by means of System Watcher's Behavioral Detection) | **Protected** |
| Hollow Process | | YES | **Protected** |
| Syscall | | YES | **Protected** |
| VBScript God Mode | | | **Protected** |
| WoW64 | | YES | **Protected** |
| Fileless detection | | | |
| Malicious PowerShell scripts | | YES | **Protected** |
| Malicious TaskScheduler tasks | | YES | **Protected** |
| WMI subscriptions | | YES | **Protected** |
| Malicious script execution via legitimate executables like mshta.exe and rundll32.exe | | YES | **Protected** |
| CVE-2013-5331 & CVE-20144113 via Metasploit | | YES | **Protected** |
| Squiblydoo AppLocker Bypass | YES | | **Protected** |
| Java Lockdown | YES | | **Protected** |
| Application Lockdown | YES | | **Protected** |

It's worth mentioning here that special attention is directed to the execution of malicious scripts, e.g. Powershell, HTA, JS/VBS, which has proved one of the most popular and dangerous techniques used in vulnerability exploitation scenarios.

Of course, some issues arise regarding mitigations provided by OS only. What about those cases when newer Operating Systems or applications supporting their embedded mitigation features can't be used? Several security vendors stress their solutions' ability to provide mitigation even in these difficult conditions - so what about Kaspersky Lab?

Our point of view here is simple: as we said earlier, we considered using these mitigations, and concluded that this just didn't pay off in the most cases. This is particularly true for legacy systems, where too many user-mode interceptions can easily consume precious system resources and slow the machine's speed below a bearable limit. And that's putting aside the fact that, as explained above, the majority of them can be sidestepped.

Even if a vulnerability in a venerable edition of MS Word is exploited, we know we'll catch it immediately afterwards anyway, when it starts behaving improperly. After all, the most important thing in exploit protection is preventing the exploit from launching the malicious payload – which is exactly what AEP does to perfection. And in the most complicated cases, as with the recent WannaCry ransomware using TCP packet-based kernel exploits, AEP, as one of the technologies leveraging Behavioral Detection, passes the baton to the next security layer, the post-execution protective mechanism. One way and another, the bottleneck is thoroughly corked.

With this sort of multi-layered approach, it's no big surprise that, as long as they had the appropriate security features switched on, Kaspersky Lab customers suffered no damage at all from the dreaded WannaCry pandemic.

Kaspersky Lab
Enterprise Cybersecurity: **www.kaspersky.com/enterprise**
Cyber Threats News: **www.securelist.com**
IT Security News: **business.kaspersky.com/**

#truecybersecurity
#HuMachine

**www.kaspersky.com**