SurfRight

HitmanPro.Alert

# Exploit Test Tool Manual

## Table of Contents

# 1 Introduction to the HitmanPro.Alert Exploit Test Tool

To be able to verify the correct working of HitmanPro.Alert we have developed an Exploit Test Tool that is able to perform specific exploit techniques. In order to use the Exploit Test Tool, make sure that HitmanPro.Alert version 3 is installed.

If HitmanPro.Alert was successfully installed and is active, every individual exploit technique test that can be started with the Test Tool should lead to an intercept message from HitmanPro.Alert. The exploit techniques performed by the Test Tool are not malicious and safe to use.

There are 2 versions of the Test Tool, a 32-bit and a 64-bit version, each with their own distinguishable icon, see Figure 1. Note that the 32-bit version of the tool contains more tests than the 64 bit version. Some of the exploit techniques are not present on 64-bit versions of the Windows operating system.

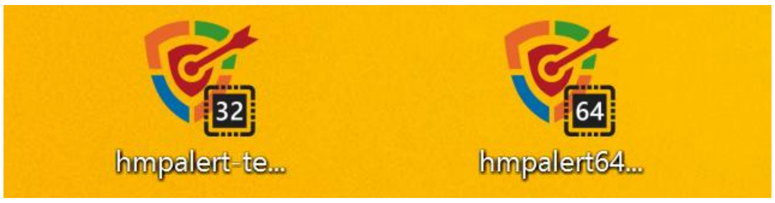
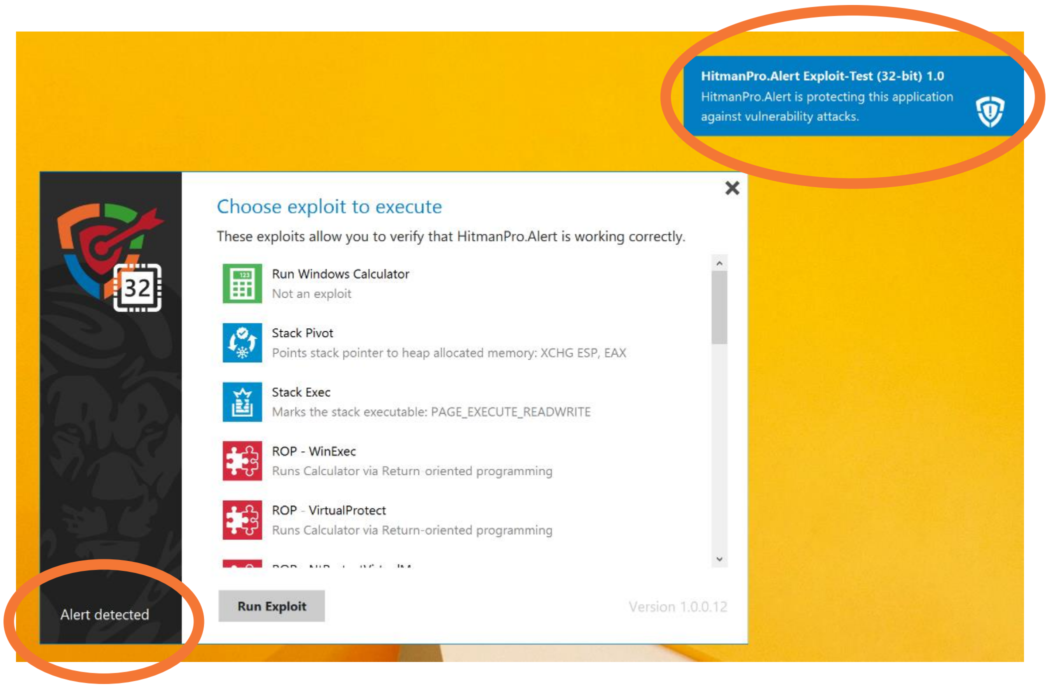


**Figure 1: 32-bit and 64-bit version of the HitmanPro.Alert Exploit Test Tool**

Once the Test Tool is started, you will see a screen like Figure 2. On the top-right part of the screen you will see a message sliding onto the desktop to indicate that HitmanPro.Alert is actively protecting the Test Tool.



HitmanPro.Alert Exploit-Test (32-bit) 1.0
HitmanPro.Alert is protecting this application against vulnerability attacks.

Choose exploit to execute
These exploits allow you to verify that HitmanPro.Alert is working correctly.

Run Windows Calculator
Not an exploit

Stack Pivot
Points stack pointer to heap allocated memory: XCHG ESP, EAX

Stack Exec
Marks the stack executable: PAGE_EXECUTE_READWRITE

ROP - WinExec
Runs Calculator via Return oriented programming

ROP - VirtualProtect
Runs Calculator via Return-oriented programming

Alert detected

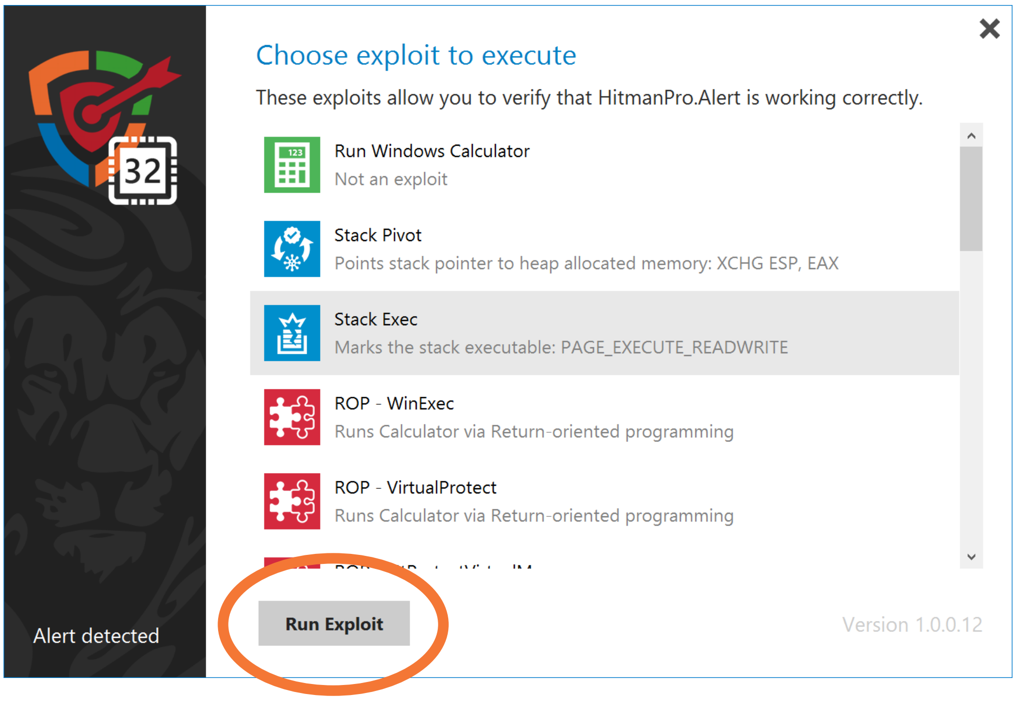
Run Exploit

Version 1.0.0.12

**Figure 2: Test selection pane**

In the left bottom corner of the Test Tool, the text **Alert detected** appears when the presence of HitmanPro.Alert is detected in the Test Tool's running process.

To start a specific test, select the test by clicking on it and then press the **Run Exploit** button to execute the exploit.



Choose exploit to execute

These exploits allow you to verify that HitmanPro.Alert is working correctly.

Run Windows Calculator
Not an exploit

Stack Pivot
Points stack pointer to heap allocated memory: XCHG ESP, EAX

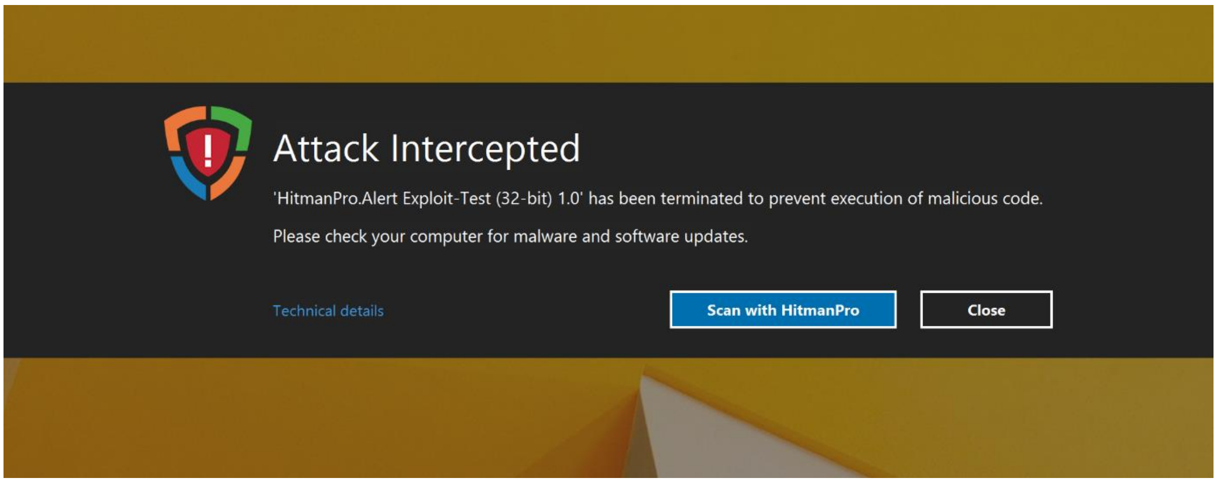
Stack Exec
Marks the stack executable: PAGE_EXECUTE_READWRITE

ROP - WinExec
Runs Calculator via Return-oriented programming

ROP - VirtualProtect
Runs Calculator via Return-oriented programming

**Figure 3: Starting an exploit test**

If an exploit technique is successfully blocked by HitmanPro.Alert, you will see an **Attack Intercepted** message as shown in Figure 4.



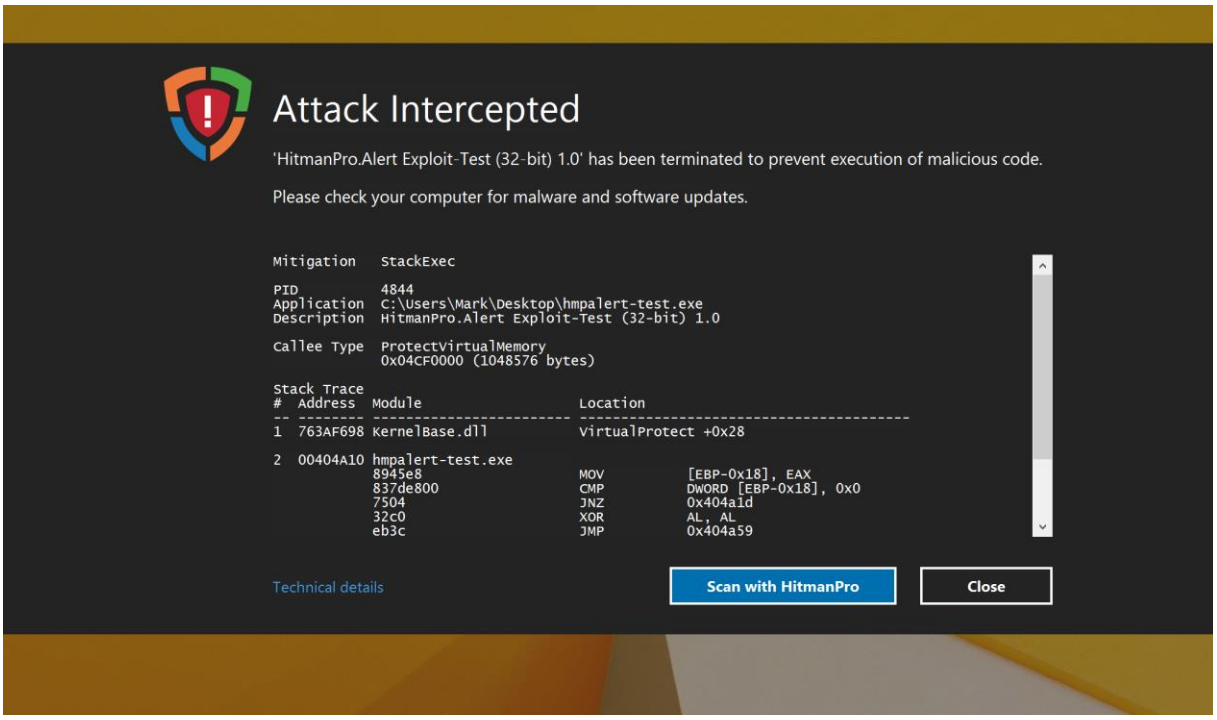
Attack Intercepted

'HitmanPro.Alert Exploit-Test (32-bit) 1.0' has been terminated to prevent execution of malicious code.

Please check your computer for malware and software updates.

**Figure 4: Attack Intercepted message**

If you click on the blue text **Technical details**, you can get more detailed information about the reason why HitmanPro.Alert generated the intercept message. A report as shown in Figure 5 will be presented.



Attack Intercepted

'HitmanPro.Alert Exploit-Test (32-bit) 1.0' has been terminated to prevent execution of malicious code.

Please check your computer for malware and software updates.

```
Mitigation   StackExec

PID          4844
Application  C:\Users\Mark\Desktop\hmpalert-test.exe
Description  HitmanPro.Alert Exploit-Test (32-bit) 1.0

Callee Type  ProtectVirtualMemory
             0x04CF0000 (1048576 bytes)

Stack Trace
#  Address  Module                    Location
-- -------- ------------------------- ----------------------------------------
1  763AF698 KernelBase.dll            VirtualProtect +0x28

2  00404A10 hmpalert-test.exe
            8945e8                    MOV          [EBP-0x18], EAX
            837de800                  CMP          DWORD [EBP-0x18], 0x0
            7504                      JNZ          0x404a1d
            32c0                      XOR          AL, AL
            eb3c                      JMP          0x404a59
```

**Figure 5: Technical details report**

## 1.1 Testing the abilities of other security software

You could also use the Exploit Test Tool to check the protection abilities of other security software, like antivirus or endpoint protection solutions designed to stop exploits or zero-day attacks. You might need to configure the third-party security software to protect the Exploit Test Tool (i.e. add the **hmpalert-test.exe** and **hmpalert64-test.exe** executables to the list of protected applications, or rename them to a filename of a known protected process e.g. **iexplore.exe**).

# 2 Background information

This section provides some background information about exploits and the different technologies that are present to counter exploitation of software vulnerabilities.

## 2.1 Blended attacks

Software exploits, commonly used by cybercriminals and nation-state hackers, are combined with computer viruses resulting in complex so-called ‘blended attacks’, which go beyond the general scope of antivirus software. To optimize success, some attackers exploit multiple vulnerabilities through several different attack vectors to silently deliver (often never before seen) malware.

Web browsers are a particular target because of their widespread distribution and usage. Attackers can also send e-mail attachments that exploit vulnerabilities in the application opening the attachment. Typically, a use-after-free or buffer overflow vulnerability in these applications can result in a call to a sensitive system function, possibly a critical function that the application was never intended to use.

## 2.2 Return-oriented programming (ROP)

Thanks to the massive adoption of Data Execution Prevention (DEP) technology, exploitation of buffer overflows through code injection is difficult; to facilitate DEP most modern processor architectures have a MMU feature called XD (eXecute Disable) or No eXecute (NX). DEP forces an application to terminate when it attempts to execute (foreign) code placed in memory areas marked for data only.

As a result, attackers now leverage existing code in the process image to compromise an application. Using existing code has been generalized in a technique called return-oriented programming (ROP), where short snippets of code – called gadgets – are chained together to introduce a new, not initially intended, control-flow. Put another way, return-oriented programming provides a fully functional "language" that an attacker can use to construct malware (shellcode) by borrowing instructions from legitimate applications running on the computer.

## 2.3 ASLR

Defenses against ROP exploits are based on randomizing the process address layout so that attackers do not know where application code is mapped in the address space. Address Space Layout Randomization (ASLR) is a technology that achieves this. It maps processes (EXEs and DLLs) in random addresses each time an application is started. Thanks to ASLR, attackers can no longer accurately predict the location of instructions that might be useful in gadgets to construct malware. Therefore, attackers now attempt to control or discover the location of certain memory regions through the use of an address space information disclosure. When you know the location of one known function, the position of all others can be inferred and a ROP attack can be constructed.

## 2.4 Control-flow integrity (CFI)

HitmanPro.Alert version 3 introduces control-flow integrity (CFI) – a technique to prevent flow of control not intended by the original application, without requiring the source code or debug symbols of the protected applications. Without requiring prior knowledge of the attack, code or malware involved, it effectively stops attackers that combine short pieces of benign code, already present in a system, for a malicious purpose (a ROP attack).

## 2.5 Hardware-assisted control-flow integrity

Whenever a critical function is triggered, HitmanPro.Alert checks whether it is a benign system call or part of a ROP exploit. It can even leverage special hardware registers inside Intel processors to assist in the detection of ROP attacks.

HitmanPro.Alert will automatically employ Intel MSR hardware registers when it e.g. detects a 2nd, 3rd or 4th generation (or newer) Core i3, i5 or i7 processor (CPU) – from 2011 and later – based on microarchitecture codenamed Sandy Bridge[1], Ivy Bridge[2], Haswell[3] (or newer); Nehalem and Westmere are not supported. E.g. desktop and mobile processors with model numbers 2xxx, 3xxx, and 4xxx (e.g. Intel Core i5-**4288**U) are supported.

To determine if CFI is supported by your specific CPU, open HitmanPro.Alert, click on the **Exploit mitigation** tile and select any application to display the exploit mitigation settings of that application. If CFI is supported by your CPU, an extra CPU icon will be visible in the **Control-Flow Integrity** box:

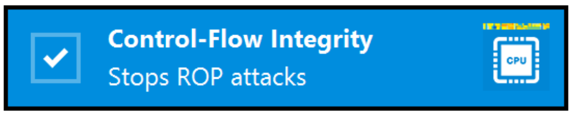
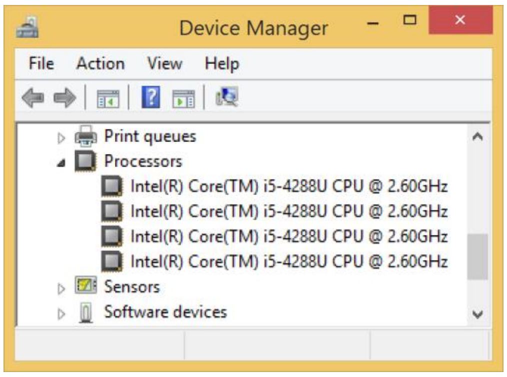


Figure 6: Control-Flow Integrity employing microprocessor hardware registers

To find the model number of your CPU, open the Windows **Control Panel** and go to **System / Device Manager**. Pay attention to the processor model number under Processors.



Print queues
Processors
Intel(R) Core(TM) i5-4288U CPU @ 2.60GHz
Intel(R) Core(TM) i5-4288U CPU @ 2.60GHz
Intel(R) Core(TM) i5-4288U CPU @ 2.60GHz
Intel(R) Core(TM) i5-4288U CPU @ 2.60GHz
Sensors
Software devices

Figure 7: Processors in Device Manager

HitmanPro.Alert will automatically fallback on software-only Control-Flow Integrity checks if your computer does not have a hardware-assisted CFI supported processor.
Note: In VMware and VirtualBox the hardware registers are not available, as these registers are not virtualized by VMware and other virtualization software.

[1] http://en.wikipedia.org/wiki/Sandy_Bridge
[2] http://en.wikipedia.org/wiki/Ivy_Bridge_(microarchitecture)
[3] http://en.wikipedia.org/wiki/Haswell_(microarchitecture)

# 3 Details of individual tests

In this section the details of the individual tests of the Exploit Test Tool will be presented.

## 3.1 Run Windows Calculator

This test is not an exploit test, but a test to verify if the tool is able to start the Windows Calculator (%systemroot%\system32\**calc.exe**). The other tests will start the Windows Calculator as proof of a successful exploit of a software vulnerability. Note that if the Windows Calculator can be started via an exploit, an attacker could have started some malicious code instead, using the exploit technique and the software vulnerability.

## 3.2 Stack Pivot

Exploit Test Tool: 32 64

A common method for an attacker to control program execution involves creating a 'fake stack' using attacker-specified values. When the attacker tricks the victim computer into using a fake stack, the attacker can control the program execution.

This specific test allocates a block of 64KB of memory on the heap and copies shellcode to that memory. Then it switches the stack pointer to the end of that memory block and starts the shellcode, which gives the shellcode a comfortably nice large stack for its operations.

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

## 3.3 Stack Exec

Exploit Test Tool: 32 64

This test makes a piece of the stack memory executable via a call to VirtualProtect. Then it copies shellcode to that part of the stack and jumps to the start of that shellcode. When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

## 3.4 ROP – WinExec

Exploit Test Tool: 32 64

This test creates a return-oriented programming (ROP) chain that calls WinExec in order to start calc.exe. After that, the ROP chain jumps to ExitProcess, so the Exploit Test Tool will end.

## 3.5 ROP – VirtualProtect

Exploit Test Tool: 32 64

This test allocates a piece of non-executable memory on the heap, where shellcode is copied to. Then it creates a ROP chain that performs the following steps:

- Call VirtualProtect to make the shellcode executable
- Jump into the shellcode, which is now executable

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

## 3.6 ROP – NtProtectVirtualMemory

Exploit Test Tool: 32 64

This test allocates a piece of non-executable memory on the heap, where shellcode is copied to. Then it creates a ROP chain that performs the following steps:

- Call NtProtectVirtualMemory to make the shellcode executable
- Jump into the shellcode, which is now executable

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

## 3.7 ROP – VirtualProtect via CALL gadget

Exploit Test Tool: 32

Important: This test is only intercepted by HitmanPro.Alert if the main processor (CPU) is a 2nd generation or newer Intel Core i3, i5 or i7 CPU (see chapter 2.5 about Hardware-assisted control-flow integrity, page 7).

This test is based on the whitepaper **Bypassing EMET 4.1**, by Jared DeMott from Bromium Labs[1].

First, it allocates a piece of non-executable memory on the heap, where shellcode is copied to. Then it locates a call to VirtualProtect in legitimate code. The address of this call is used in the ROP chain.

Then it creates a ROP chain that performs the following steps:

- Jump to the legitimate code where a call to VirtualProtect is located
- Jump into the shellcode, which is now executable

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

[1] http://labs.bromium.com/2014/02/24/bypassing-emet-4-1/

## 3.8 NULL Page

Exploit Test Tool: 32 64

This test allocates the first page of the virtual memory of the process and copies shellcode into that page. Then it jumps to the shellcode, thus simulating the (faulty) situation where a pointer is used to call a function and that pointer is NULL because of some unexpected condition. When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

## 3.9 SEHOP

Exploit Test Tool: 32

HitmanPro.Alert includes Structured Exception Handler Overwrite Protection (SEHOP), which currently protects against the most common technique for exploiting stack overflows in Windows.

The SEHOP test in the Test Tool creates a buffer overflow, which causes the overwriting of the Windows Structured Exception Handler (SHE) record that is located on the stack and also puts our shellcode on the stack. It overwrites the SEH record with a pointer to a known 'POP-POP-RET' sequence and a short jump instruction to the shellcode that is now on the stack. Then it generates an exception which causes the exception handler to traverse the SEH chain and in doing so executing our shellcode on the stack. When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

Note: To let this test start calc.exe, both the mitigations **Stack Exec** and **SEHOP** must be disabled.

## 3.10 Heap Spray 1 – Single byte NOP-sled

Exploit Test Tool: 32 64

Heap spraying[1] is a technique used in exploits to facilitate arbitrary code execution. The Dynamic Heap Spray protection in HitmanPro.Alert does not pre-allocate common heap spray addresses but is a generic solution aimed to detect heap spray behavior.

This specific Heap Spray test in our Exploit Test Tool allocates 256MB of memory on the heap and stamps this memory from begin to end with a 64KB template. This template consists for the largest part of a single-byte NOP-sled[2], followed by a small shellcode. Once the complete 256MB of memory is filled, the region is made executable via a call to VirtualProtect and the test jumps into a NOP-sled in that memory range which leads to the start of the shellcode. When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

Note: To let this test start calc.exe, **Control-Flow Integrity** and **Dynamic Heap Spray** must be disabled.

[1] http://en.wikipedia.org/wiki/Heap_spraying

[2] http://en.wikipedia.org/wiki/NOP_sled

## 3.11 Heap Spray 2 – Polymorphic NOP-sled

Exploit Test Tool: 32 64

This test allocates 256MB of memory on the heap and stamps this memory from begin to end with a 64KB template. This template consists for the largest part of a multi-byte NOP-sled (polymorphic NOP-sled), followed by a small shellcode. Once the complete 256MB of memory is filled, the region is made executable via a call to VirtualProtect and the test jumps into a NOP-sled in that memory range which leads to the start of the shellcode.

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

Note: In order to let this test successfully start calc.exe, both the mitigations **Control-Flow Integrity** and **Dynamic Heap Spray** must be disabled.

## 3.12 Heap Spray 3 – ActionScript

Exploit Test Tool: 32 64

This test allocates 16MB of memory on the heap and stamps this memory from begin to end with a 1024 byte template. This template consists of an Adobe ActionScript vector of UInts (used by Adobe Flash), simulating the actions performed by e.g. CVE-2014-0322[1] and CVE-2014-1776[2].

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

## 3.13 Heap Spray 4 – JavaScript

Exploit Test Tool: 32 64

This test allocates 16MB of memory on the heap and stamps this memory from begin to end with a 64KB template. This template consists of a JavaScript ArrayBuffer object, simulating the action as is performed by e.g. CVE-2014-1512[3].

When calc.exe has been started, the Exploit Test Tool terminates via a call to ExitProcess.

[1] http://www.fireeye.com/blog/technical/cyber-exploits/2014/02/operation-snowman-deputydog-actor-compromises-us-veterans-of-foreign-wars-website.html

[2] http://www.fireeye.com/blog/uncategorized/2014/04/new-zero-day-exploit-targeting-internet-explorer-versions-9-through-11-identified-in-targeted-attacks.html

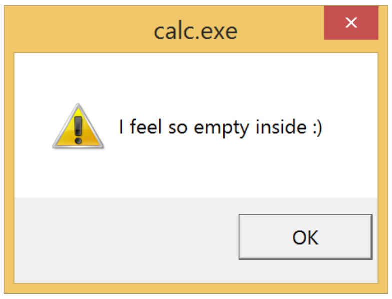
[3] http://www.cvedetails.com/cve/CVE-2014-1512/

## 3.14 Hollow Process

Exploit Test Tool: 32

Process hollowing is a technique used by some malware (in many Remote Access Trojans) in which a legitimate process is loaded on the system solely to act as a container for hostile code.

This specific test works on **32-bit operating systems only** and starts the Windows Calculator calc.exe from your Windows installation. It then replaces the in-memory contents of that process with a small program that displays the following message box.


calc.exe

I feel so empty inside :)

Figure 8: Calc.exe process abused to run arbitrary code

Note: In order to let this test successfully start and manipulate the binary contents of the calc.exe process, both the code mitigations **Deny New Process** and the System Security function **Hollow Process** must be disabled.

## 3.15 Load Library

Exploit Test Tool: 32 64

This test tries to download a library file (DLL file) over a UNC network path, a common technique used by attackers.

## 3.16 URLMon

Exploit Test Tool: 32 64

This test allocates a piece of executable memory on the heap, where shellcode is placed. This shellcode is then started.

The shellcode then calls the system function URLDownloadToFileA to retrieve the remote file http://test.hitmanpro.com/dummy.dll

When the download was successful it will start calc.exe. In an attack scenario the download is a malicious payload to infect the computer.

## 3.17 URLMon 2 – Rundll32

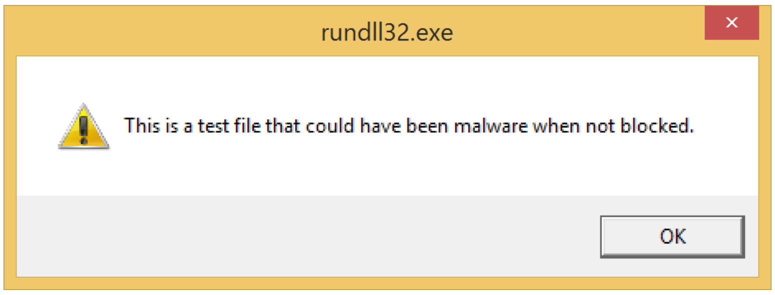
Exploit Test Tool: 32

This test creates a ROP chain that performs the following steps:

- Call the system function URLDownloadToFileA to retrieve the remote file: http://test.hitmanpro.com/dummy.dll

- Load the downloaded dummy.dll using **Rundll32**

If the start of the ROP chain is successful you will see a message box as shown below. This message is generated by the downloaded DLL that could successfully execute.


rundll32.exe

This is a test file that could have been malware when not blocked.

In an attack scenario the download is a malicious payload to infect the computer.

## 3.18 URLMon 3 – LoadLibraryA

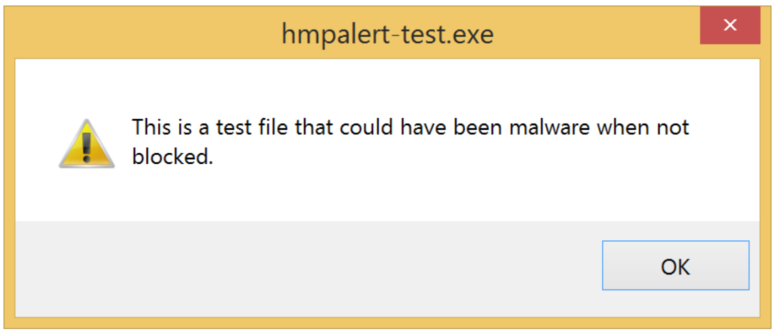
Exploit Test Tool: 32

This test creates a ROP chain that performs the following steps:

- Call the system function URLDownloadToFileA to retrieve the remote file: http://test.hitmanpro.com/dummy.dll

- Load the downloaded dummy.dll using the system function **LoadLibraryA**

If the start of the ROP chain is successful you will see a message box as shown below. This message is generated by the downloaded DLL that could successfully execute.


hmpalert-test.exe

This is a test file that could have been malware when not blocked.

In an attack scenario the download is a malicious payload to infect the computer.

## 3.19 Webcam test (not an exploit)

Exploit Test Tool: 32 64

This test is not an exploit test, but a way to verify if the Webcam Notifier functionality of HitmanPro.Alert is operating correctly. When this test is started, a window will appear which shows the captured video from your webcam. If the Webcam Notifier function is enabled in HitmanPro.Alert you will see a red message sliding onto the desktop to notify that the webcam is currently active and is capturing video.

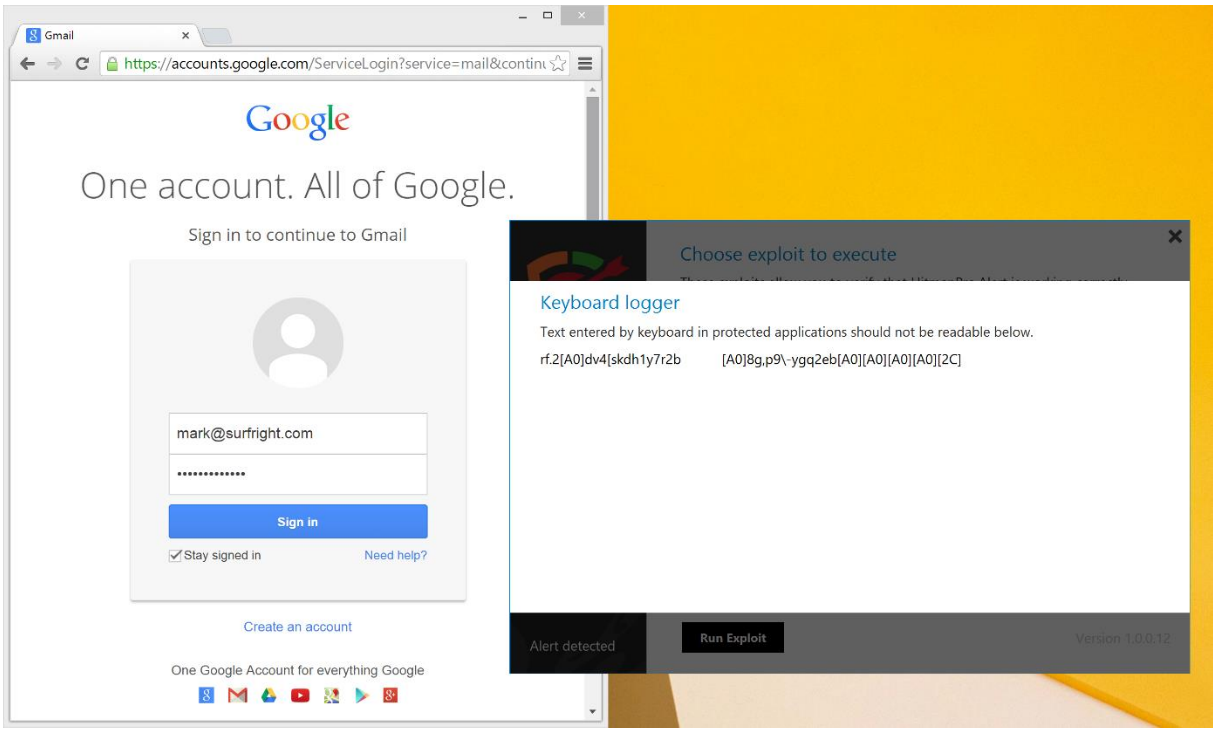
## 3.20 Keyboard logger (not an exploit)

Exploit Test Tool: 32 64

This test is not an exploit test, but a way to verify if the Keyboard Encryption functionality of HitmanPro.Alert is operating correctly. Note that the Keyboard Encryption functionality in HitmanPro.Alert is applied to web browsers only.

If this test is started, you will see a box overlaying the Test Tool window. In this text box you can see the keystrokes an attacker would see when he tries to intercept your keystrokes. To see what the effect is, you can open a web browser and type some text, for example on a web page where login credentials are requested.

When the Keyboard Encryption function is enabled, you will see random characters in the text box of the Test Tool. See Figure 9 for an example.


Gmail
https://accounts.google.com/ServiceLogin?service=mail&continu
Google
One account. All of Google.
Sign in to continue to Gmail
mark@surfright.com
Sign in
Stay signed in
Need help?
Create an account
One Google Account for everything Google
Choose exploit to execute
Alert detected
Run Exploit
Version 1.0.0.12

Keyboard logger

Text entered by keyboard in protected applications should not be readable below.

```
rf.2[A0]dv4[skdh1y7r2b        [A0]8g,p9\-ygq2eb[A0][A0][A0][A0][2C]
```

**Figure 9: Keyboard Encryption activated**

## Revision History

| Version | Authors | Remarks | Date |
|---|---|---|---|
| 1.0 | EE, ML | Initial version | 2014-7-7 |