AN OLD HILLBILLY'S GUIDE TO BASH FOR PENTESTS

Automation, Logging, and Covering your Butt

WHAT YOU SHOULD ALREADY KNOW (*OR PRETEND TO*)

Familiarity with Linux/Unix command-line interfaces.

Basic experience with shell environments (BASH, ZSH, SH).

General understanding of scripting concepts.

WHAT EXACTLY IS BASH, AND WHY SHOULD I CARE?

Bourne Again Shell (BASH): Enhanced version of the original Unix Bourne Shell (sh).

Default command-line shell on Linux systems.

Powerful scripting capabilities for automation and system management.

WHAT BASH CANNOT (AND SHOULD NOT) DO

Not a general-purpose programming language (limited data structures).

Unsuitable for complex GUI or performance-intensive applications.

Limited error-handling compared to languages like Python or Ruby.

WHERE IN THE WORLD IS BASH?

- Linux (default shell).
- macOS (default until recent shift to Zsh).
- Windows via WSL, Git Bash, Cygwin.
- Embedded devices and IoT.

WHY BOTHER WITH BASH SCRIPTING IN PENETRATION TESTING?

What is BASH Scripting?

- Automating sequences of command-line tasks.
- Streamlining repetitive and tedious security-testing processes (e.g., scanning, enumeration, exploitation attempts).

Why use BASH in Penetration Testing?

- **Efficiency:** Drastically reduces manual effort by automating routine procedures.
- **Consistency:** Ensures uniform execution of tests, minimizing variations caused by human oversight.
- **Accuracy:** Reduces the likelihood of errors through automated command execution.
- **Documentation:** Enhances detailed logging, ensuring a clear audit trail for reporting and compliance.

THE SIMPLEST BASH SCRIPT YOU WILL EVER WRITE

```
#!/bin/bash
echo "Hello, World!"
```

```
#!/usr/bin/env bash
echo "Hello, World!"
```

THE BUILDING BLOCKS OF BASH AUTOMATION

- Shebang (#!/bin/bash): Defines script interpreter.
- Variables: Store reusable information.
- Functions: Encapsulate reusable code blocks.
- Loops: Automate repetitive tasks (for, while).
- File I/O: Read and write data files.

AUTOMATING SCANS WITH A SIMPLE LOOP

```
while read -r ip; do
    nmap --open "$ip"
done < ips.txt
```

EXTRACTING MEANING FROM COMMAND OUTPUT

```
#!/usr/bin/env bash
target="$1"
tempfile=$(mktemp)
nmap -oG "$tempfile" "$target" > /dev/null

echo "Open ports on $target:"
grep "Ports:" "$tempfile" \
  | grep "/open" \
  | sed -n 's/.*Ports: \(.*\)/\1/p' \
  | tr ',' '\n' \
  | grep "/open" \
  | awk -F/ '{print $1} '

rm -f "$tempfile"
```

LOGGING AS YOU GO: THE POWER OF TEE

```
#!/bin/bash
domain="example.com"
log_dir="logs"
mkdir -p "$log_dir"

whois "$domain" | tee "$log_dir/whois_$domain.txt"
```

TRACKING EVERYTHING YOU TYPE (ON PURPOSE)

```
#!/usr/bin/env bash
if [[ -n "$TMUX" ]]; then
  session_name="$(tmux display-message -p '#S' 2>/dev/null ||
  echo "tmux_$$")"
  export HISTFILE="$HOME/.bash_history_tmux_${session_name}"
elif [[ -n "$STY" ]]; then
  session_name="${STY#*.}"
  export HISTFILE="$HOME/.bash_history_screen_${session_name:-
  $$}"
else
  export HISTFILE="$HOME/.bash_history"
fi
```

TRACKING EVERYTHING YOU TYPE (ON PURPOSE)

```
export HISTCONTROL=
export HISTIGNORE=
export HISTSIZE=100000
export HISTFILESIZE=100000
export HISTTIMEFORMAT="[%m/%d/%y %H:%M:%S] "
shopt -s histappend
```

TRACKING EVERYTHING YOU TYPE (ON PURPOSE)

```
if [[ ! -f "$HISTFILE" ]]; then
  touch "$HISTFILE"
  chmod 600 "$HISTFILE"
fi
```

TRACKING EVERYTHING YOU TYPE (ON PURPOSE)

```
history_sync_cmd="history -a; history -n; history -w"

if [[ -z "$PROMPT_COMMAND" ]]; then
  PROMPT_COMMAND="$history_sync_cmd"
elif [[ "$PROMPT_COMMAND" != *"$history_sync_cmd"* ]]; then
  PROMPT_COMMAND="${PROMPT_COMMAND%;}; $history_sync_cmd"
fi
```

BASH AUTOMATION: NMAP -> METASPLOIT

Goal: Run a repeatable MSSQL brute-force test

Tools: Metasploit + Bash + Nmap output

Features:

- Uses known-good wordlists
- Pulls filtered targets from .gnmap files
- Outputs results to logs for report inclusion

THE METASPLOIT RC FILE

```
<ruby>
module_path = "auxiliary/scanner/mssql/mssql_login"
user_file = "/opt/default_lists/mssql_users.txt"
pass_file = "/opt/default_lists/mssql_passwords.txt"
rhosts_file = "/root/1433.txt"
...
self.run_single("use #{module_path}")
self.run_single("set USER_FILE #{user_file}")
self.run_single("set PASS_FILE #{pass_file}")
self.run_single("set RHOSTS file:#{rhosts_file}")
self.run_single("run")
...
</ruby>
exit
```

SESSION MANAGEMENT AND OUTPUT LOGGING

```
...
tee_dir = "/root/TEE"
tee_log_path = "#{tee_dir}/MSF_mssql_login.tee"

self.run_single("spool #{tee_log_path}")
...
self.run_single("spool off")
...
```

BUILDING THE RHOSTS LIST WITH BASH

```
#!/usr/bin/env bash
output_file="/root/1433.txt” > "$output_file"

find . -name "*.gnmap" | while read -r file; do
  grep "1433/open/tcp" "$file" | awk '{print $2}' >> "$output_file"
done

sort -u -o "$output_file" "$output_file"
```

HOW IT ALL CONNECTS

Nmap Scan ➜ *.gnmap

↓

build_1433_list.sh

↓

/root/1433.txt

↓

mssql_login.rc

↓

Metasploit Autoscans

↓

/root/TEE/

TRIGGER IT ALL FROM BASH

```
build_1433_list.sh /scans/project1
msfconsole -q -r /root/mssql_login.rc | tee "logs/msf_$(date +%F).log"
```

WHY TO CUSTOMIZE THE BASH PROMPT?

Your prompt is your heads-up display.

Know your environment at a glance:

- Time and date
- Session name (tmux, screen)
- Kerberos ticket presence
- Git repo context
- Username, hostname, and current directory

BASH PROMPT: EXAMPLE OUTPUT

```
[04/16/25 11:04:33] [Screen: Recon] [krb5cc: admin.ccache] [git: tools]
root@attacker:/root/ #
```

BASH PROMPT: DATE AND SESSION

```
local datetime="\[\e[36m\][\D{%m/%d/%y %H:%M:%S}]\[\e[0m\] "

if [[ -n "$TMUX" ]]; then
  session_name=$(tmux display-message -p '#S')
  session="\[\e[33m\][tmux: ${session_name}]\[\e[0m\]"
elif [[ -n "$STY" ]]; then
  session_name="${STY#*.}"
  session="\[\e[33m\][screen: ${session_name}]\[\e[0m\]"
fi
```

BASH PROMPT: KERBEROS AND GIT

```
if [[ -n "$KRB5CCNAME" ]]; then
  krb5cc="\[\e[32m\][krb5cc: present]\[\e[0m\]"
fi

if git rev-parse --is-inside-work-tree &>/dev/null; then
  repo_name=$(basename "$(git rev-parse --show-toplevel)")
  gitrepo="\[\e[35m\][git: ${repo_name}]\[\e[0m\]"
fi
```

BASH PROMPT: PUTTING IT ALL TOGETHER

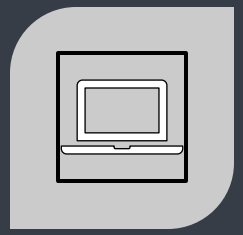
```
PS1="${datetime} ${session} ${krb5cc} ${gitrepo}\n\u@\h:\w \$ "

PROMPT_COMMAND=custom_pentest_prompt
```

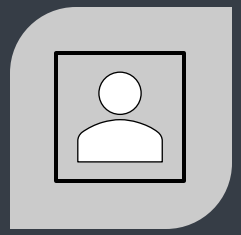
BASH PROMPT:

BONUS IDEAS

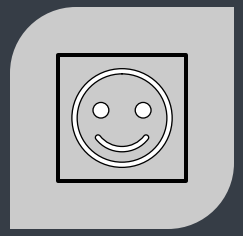
SHOW DIFFERENT PROMPTS FOR ROOT VS NON-ROOT



DISPLAY CURRENT EXTERNAL IP OR LOCAL INTERFACE



SHOW ACTIVE PYTHON VENV OR AWS PROFILE



ADD EMOJI IF YOU HATE HAPPINESS

DO YOU REALLY NEED TO SCRIPT? (SHORT ANSWER: YES)

Automation in Penetration Testing:

- Penetration testing involves repetitive tasks (scanning, enumeration, exploitation).
- Automation significantly reduces human error and saves time.

Role of AI Tools:

- Increased availability of AI-generated scripts (ChatGPT, Copilot).
- Scripts provided by AI are helpful but rarely flawless.

WHY AI STILL NEEDS YOU TO KNOW BASH

AI-generated scripts:

- Fast generation, useful starting points.
- Frequent minor mistakes (syntax, logic, compatibility).

Human scripting skills:

- Essential for troubleshooting and debugging.
- Allow customization to specific environments or targets.
- Critical for adapting to unforeseen issues during assessments.

PRACTICAL TIPS FOR BLENDING AI WITH BASH

01

Start with AI-generated scripts but always verify and debug.

02

Develop strong fundamentals in BASH scripting.

03

Balance automation with human insight for optimal outcomes.

EFFICIENCY AND ACCURACY: YOUR NEW BEST FRIENDS

Efficiency:

- Automates repetitive and tedious tasks.
- Saves significant manual effort.

Accuracy:

- Minimizes human error in repetitive processes.
- Ensures consistent execution across multiple assessments.

LOGS AND REPEATABILITY: PROOF YOU DID THE WORK

Structured Logging:

- Detailed and structured logs for reporting and auditing.
- Essential for documenting findings and actions.

Repeatability:

- Scripts provide uniform results across similar tests.
- Easy to reuse, adapt, and improve over time.

KEEP GROWING: SCRIPTING HABITS THAT SCALE

Practice regularly:

Regularly refine and improve scripts.

Stay updated with new techniques and tools.

Build a script library:

Develop a personal or organizational library of reusable scripts.

Continually expand your toolkit for future engagements.

YOUR TURN: QUESTIONS, SCENARIOS, AND SHOW-AND-TELL

COMMON QUESTIONS AND SCRIPT SUGGESTIONS

"What resources do you recommend to learn more?"

"Can you provide example scripts or repositories?"

"How do you securely manage logs and outputs during penetration tests?"

Thank You for Attending!

Presenter Information:

- Name: **Adam Compton**
- Title: **Principal Security Consultant**
- Organization: **TrustedSec**
- Email: **adam.compton@trustedsec.com**
- Twitter: **@tatanus**

STAY IN TOUCH— LET'S KEEP SCRIPTING