SOC Playbook: Bash Script Abuse Detection (T1059.004)

1. Objective

Detect and respond to suspicious or malicious use of **Bash scripts**, including unauthorized execution, obfuscated payloads, or misuse for persistence, lateral movement, or data exfiltration.

2. Scope

- Detect suspicious Bash activity across Linux/macOS environments.
- Identify misuse of Bash for running encoded, dropped, or backdoored scripts.
- Detect malicious behavior from cron jobs, SSH sessions, or web shells.
- Respond to fileless threats or post-exploitation via Bash.

3. Log Sources

Platform	Log Source	Description
Linux/macOS	Syslog / Auditd (execve syscall)	Tracks command execution
Linux/macOS	.bash_history, .zsh_history	Historical command context
Linux/macOS	OSQuery	Real-time process and file monitoring
All	EDR / XDR	Process telemetry and alert correlation
All	File Integrity Monitoring (e.g. Tripwire)	Detects changes to script or config files

4. Detection Rules / Alerts

Alert Name	Description	Conditions / Triggers
Encoded Bash Commands	Use of base64, eval, or pipe chaining	Bash using base64, eval $(...), `
Bash from Web Directory	Script execution from /var/www/, /tmp/	Executed from unusual or public paths
Hidden/Obfuscated Scripts	Scripts with . prefix, strange names	.xyz.sh, long base64 blob names, etc.
Cron Job Abuse	Malicious persistence via scheduled tasks	Unexpected cron job entry creating Bash invocation
Reverse Shell via Bash	Use of Bash to open network connections	bash -i >& /dev/tcp/, nc -e /bin/bash
Script via Web Shell	Bash command executed by web server or PHP binary	Parent process is apache, nginx, php-fpm

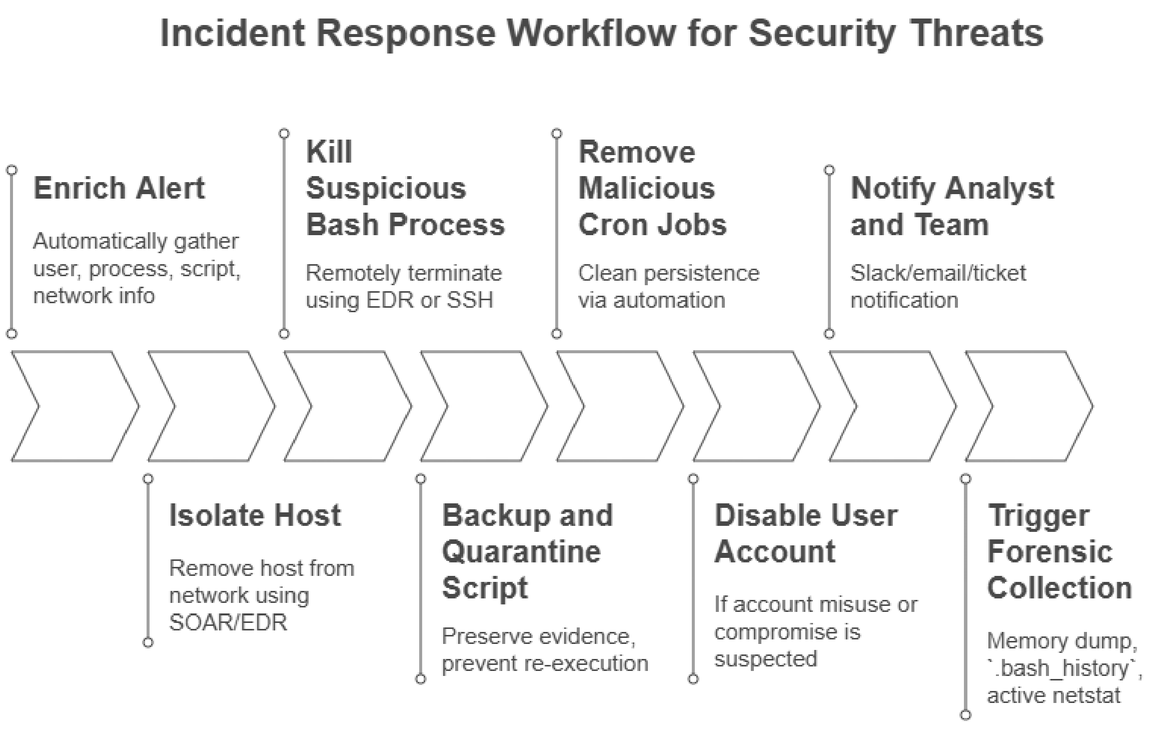
SSH Tunneling with Bash	Malicious script used after SSH session	Interactive Bash session with lateral movement

5. Automated Enrichment

Enrichment Task	Details
User Attribution	Identify login user, shell environment
Process Lineage	Trace parent-child process chain via ps, auditd
Script Content	Pull and analyze script content using OSQuery or EDR
Network Analysis	Correlate connections opened by the script (NetFlow, Zeek)
File Hash Lookup	Run hashes through VirusTotal or internal DB

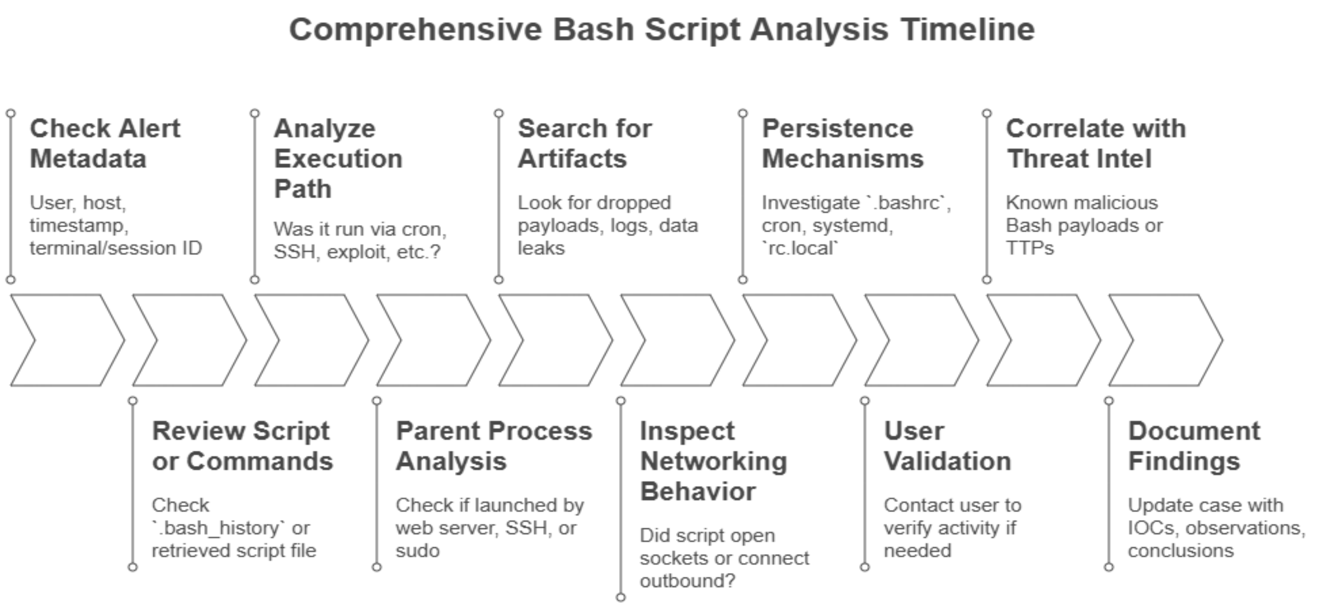
6. Automated Response Play

Step	Action
1. Enrich Alert	Automatically gather user, process, script, network info
2. Isolate Host	Remove host from network using SOAR/EDR
3. Kill Suspicious Bash Process	Remotely terminate using EDR or SSH
4. Backup and Quarantine Script	Preserve evidence, prevent re-execution
5. Remove Malicious Cron Jobs	Clean persistence via automation
6. Disable User Account	If account misuse or compromise is suspected
7. Notify Analyst and Team	Slack/email/ticket notification
8. Trigger Forensic Collection	Memory dump, .bash_history, active netstat


Incident Response Workflow for Security Threats
Enrich Alert
Automatically gather user, process, script, network info
Isolate Host
Remove host from network using SOAR/EDR
Kill Suspicious Bash Process
Remotely terminate using EDR or SSH
Backup and Quarantine Script
Preserve evidence, prevent re-execution
Remove Malicious Cron Jobs
Clean persistence via automation
Disable User Account
If account misuse or compromise is suspected
Notify Analyst and Team
Slack/email/ticket notification
Trigger Forensic Collection
Memory dump, `.bash_history`, active netstat

7. Investigation Checklist

Step	Description
1. Check Alert Metadata	User, host, timestamp, terminal/session ID
2. Review Script or Commands	Check .bash_history or retrieved script file
3. Analyze Execution Path	Was it run via cron, SSH, exploit, etc.?
4. Parent Process Analysis	Check if launched by web server, SSH, or sudo
5. Search for Artifacts	Look for dropped payloads, logs, data leaks
6. Inspect Networking Behavior	Did script open sockets or connect outbound?
7. Persistence Mechanisms	Investigate .bashrc, cron, systemd, rc.local
8. User Validation	Contact user to verify activity if needed
9. Correlate with Threat Intel	Known malicious Bash payloads or TTPs
10. Document Findings	Update case with IOCs, observations, conclusions


Comprehensive Bash Script Analysis Timeline
Check Alert Metadata
User, host, timestamp, terminal/session ID
Review Script or Commands
Check `.bash_history` or retrieved script file
Analyze Execution Path
Was it run via cron, SSH, exploit, etc.?
Parent Process Analysis
Check if launched by web server, SSH, or sudo
Search for Artifacts
Look for dropped payloads, logs, data leaks
Inspect Networking Behavior
Did script open sockets or connect outbound?
Persistence Mechanisms
Investigate `.bashrc`, cron, systemd, `rc.local`
User Validation
Contact user to verify activity if needed
Correlate with Threat Intel
Known malicious Bash payloads or TTPs
Document Findings
Update case with IOCs, observations, conclusions

8. Playbook Notes

- Restrict script execution permissions (chmod, sudo restrictions).
- Monitor for hidden or base64-encoded .sh files.
- Configure audit rules to monitor execve and /bin/bash usage.
- Enable bash_command_audit in newer Linux kernels (where available).
- Capture .bash_history changes using FIM or OSQuery.
- Watch for command chains like: curl | bash, wget | bash.