Are We Affected and Where? | Software Inventory

The process of software profiling involves understanding what exists in your environment and how trustworthy it is. The practices of hashing and code signing are commonly used to understand authenticity and integrity, but may not be sufficient for high assurance if the cryptographic supply chain has been compromised as well.

Software Inventory Profiling (Windows Powershell)

`Get-ChildItem -Path "C:\Program Files\" -Recurse -Include "*.exe"`	Search common paths for executable. Modify path or extension as needed.
`Get-WmiObject -Query "SELECT * FROM Win32_Product" | Select-Object Name, Version, InstallDate, InstallLocation, Vendor`	Using WMI to look for installed software and obtain vendor, software name, version and installation path which can be used for further analysis.
`Get-AuthenticodeSignature -FilePath "C:\Path"`	Check code signing—it can also be used to verify powershell scripts.
`Get-FileHash -Path "C:\Path " -Algorithm MD5`	Generate Hash (MD5, SHA1, SHA256, etc)

OS File Verification (Windows)

`Sigverif.exe`	Verify integrity of core Windows OS files and signing.

Certificate Verification (Windows)

`certutil -dump "C:\Path"`	Use certificate utility to dump details of signed binary.
`Third-Party Tools`	Openssl, sigcheck

Software Inventory Profiling (Linux)

`apt list --installed`	Apt based package manager (apt cache policy for more information)
`snap list`	Snap installed software
`dpkg --list`	Dpkg packaga manager on Debian (dpkq-query for more information)
`rpm -qa`	Red Hat based Linux variants
`yum list installed`	Yum package manager
`find /usr/bin/ -type f -exec ls -l {} \;`	Generic path checks for installed executables, change path for other directories
`File "file"`	Determine executable type, architecture, other metadata
`md5sum | sha1sum | sha256sum`	Generate Hash (MD5, SHA1, SHA256, etc)
`openssl dgst -sha256 -verify <public_key.pem> -signature <signature.sig> <file>`	Check signatures using openssl if you have a detached certificate
`apt-key list`	List installed gpg keys
`lsmod and modinfo <modulename>`	Verify kernel modules are signed
`Jarsigner --verify <jarfile>`	Verify signed jar files

Additional Code Signing Tools

`cosign verify <image>`	In addition to binaries and container images, Cosign supports transparency logs, detached signatures, keyless signing and more
`gpg –verify <signature> <file>`	Verify with detached signature, or import with
`gpg –import <keyfile>`	Import public key used in signing

You may be looking at a software distribution channel that has been compromised, and not a specific package.

How do We Stop the Bleeding? [Containment]

Contain the Threat (Windows Powershell)

`New-NetFirewallRule -DisplayName "Block Malicious IP" -Direction Outbound -RemoteAddress <malicious IP> -Action Block`	Block a malicious IP address by creating a firewall policy
``Add-Content -Path "C:\Windows\System32\drivers\etc\hosts" -Value "`n127.0.0.1 <malicious domain>`n"``	Block malicious domains with a DNS blackhole
`Get-AppLockerPolicy -Effective`	List currently applied AppLocker policies

Contain the Threat (Linux)

`iptables -A INPUT -s <malicious IP> -j DROP`	Block a malicious IP address with iptables firewall rule
`echo "127.0.0.1 <malicious domain>" | sudo tee -a /etc/hosts`	Block block malicious domain with a DNS blackhole

Supporting Commands

Supporting System Commands (Linux)

`curl -s <URI>`	Command line access to work with APIs and web requests. JSON responses can be processed with jq (below)
`jq`	Useful for JSON parsing
`file`	Determining file attributes, executable architecture and more
`strings`	Extract strings from files, including component names, comments, etc.
`grep`	Search patterns for specific strings, regular expressions
`find`	Locate a specific file

Working with SBOM JSON Files

`jq '.components[] | {name, version}' sbom.json`	List all components
`jq '.components[] | select(.name = "component_name")' sbom.json`	Find specific components
`jq '.components[] | select(.purl = "specific_purl")' sbom.json`	Find specific PURL
`jq '.components[] | select(.purl = "specific_purl" and .version = "specific_version")' sbom.json`	Combine PURL with version
`jq '.components[] | {name, licenses}' sbom.json`	Check licenses
`jq '.components[] | select(.type = "library")' sbom.json`	Filter by component type

Profiling Docker Containers

`docker ps`	List running Docker containers
`docker images`	List downloaded Docker images
`docker inspect <image_id>`	Details on a specific image id
`docker logs <container_id>`	Shows logs for a running container

Software Profiling Package Managers

`pip list |  pip3 list`	List installed pypi packages
`cpan -l`	List installed cpan modules
`cargo install --list`	List installed cargo packages
`npm list -g`	List globally installed npm packages
`ls $(go env GOPATH)/bin`	List all installed go packages by specifying the GOBIN variable. Normally ~/go/bin

SANS
CYBER DEFENSE

Software Supply Chain Incident Response Cheat Sheet v1.0

This guide was created by **Tony Turner**
X: @tonylturner | sans.org/sec547

The purpose of this reference guide is to provide rapid access to useful commands helpful when responding to a software supply chain security incident.

This cheat sheet assumes that you may have been caught unprepared by a supply chain security incident, and focuses on Identification, Containment and Eradication phases of the SANS PICERL incident response methodology. As such, we have summarized in the following three activities:

1. Are we affected and where?
2. What is the impact?
3. How do we stop the bleeding?

It is assumed that when starting your incident response, that you have been alerted to a potential issue that will focus your efforts. Some resources below that may help you get started.

Supply Chain Security Incident Resources

Risk Explorer for Software Supply Chain
https://sap.github.io/risk-explorer-for-software-supply-chains/

Open Software Supply Chain Attack Reference (OSC&R)
https://pbom.dev/

MITRE ATT&CK Supply Chain Compromise
https://attack.mitre.org/techniques/T1195/

OSV Vulnerability Database
https://osv.dev/

National Vulnerability Database
https://nvd.nist.gov/

Dataset of 184 supply chain attacks
https://github.com/IQTLabs/software-supply-chain-compromises

Atlantic Council SSC dataset and visualization
https://www.atlanticcouncil.org/commentary/trackers-and-data-visualizations/breaking-trust-the-dataset/

Disclaimer:
We further assume that the focus is on returning to normal operation, not prosecution. As such, these techniques may not be forensically sound from an evidence preservation standpoint. Always consult with your legal team and internal incident response processes to determine suitability in your environment.

Are We Affected and Where? | Indicators of Compromise

Investigating system and application logs

Event Viewer (Windows Powershell)

Get-WinEvent -LogName Application | Where-Object { $_.Id -eq 1033 }	MSI Installs and uninstalls: • Event ID 1033: Software installed • Event ID 1034: Software uninstalled • Event ID 11707: Success of an application installation. • Event ID 11708: Failure of an application installation.
Get-WinEvent -LogName Application | Where-Object { $_.ProviderName -eq "MsiInstaller" -and $_.Id -eq 1034 } | ForEach-Object { $_ \| Select-Object TimeCreated, Id, ProviderName, @{Name="Message" ;Expression={($_.Message -join "`n")}} }	The command above will include other provider names and truncate output, you can expand on that by limiting to MsiInstaller and building a query with the specific information you are looking for.
Get-WinEvent -LogName "System" | Where-Object { $_.Id -eq 19 } | Format-Table TimeCreated, Message	For Windows Updates: • Event ID 19: Successful installation • Event ID 20: Failed installation • Event ID 21: Update is available
Get-WinEvent -LogName "Security" | Where-Object { $_.Id -eq 4688 }	Security logs worth investigating: • Event ID 4688: Process Creation • Event ID 4624: Successful Account Logon • Event ID 4625: Failed Account Logon • Event ID 4720: User Account Created • Event ID 4672: Privileged Logon • Event ID 4663: File Creation

Syslog (Linux)

grep <package> /var/log/syslog	Search syslog for mention of a package
grep -E '(install|upgrade|remove|purge) ' /var/log/dpkg.log	Search dpkg log, using -E for regular expressions to look for specific installation or upgrade mentions on Debian systems
grep -E 'Installed:|Updated:' /var/log/yum.log	Search yum log, using -E for regular expressions to look for specific installation or update mentions on Red Hat systems

Note: Consider enabling DNS logging locally or interrogating internal DNS forwarders and firewalls to look for indicators of known bad domains. You can also interrogate the DNS cache.

DNS Cache (Windows)

ipconfig /displaydns | Select-String -Pattern "<domain>"	Using Windows native ipconfig and powershell

YARA is a pattern-matching tool used to detect malware and suspicious artifacts in files, binaries, or memory by writing rules that define specific strings or behaviors. Example: Identifying suspicious imports in firmware binaries or specific signatures.

```
rule SuspiciousLibrary
{
    strings:
        $lib1 = "backdoor.dll"
        $lib2 = "malicious.dll"
    condition:
        $lib1 or $lib2
}
```

Yara String Options

$var_name = "string"	Test strings
$var_name = { 6A 40 68 ?? ?? 6A 14 8D }	Hexadecimal strings may be more accurate for binaries, ?? are wildcard
$regex_pattern = /malware[0-9]{3}/	Regular expressions
$case_insensitive_str = "maliciouscode" nocase	Case insensitive strings
$wide_string = "malware_signature_here" wide	Wide strings match UTF-16 encoding
$b64_str = "malware_signature_here" base64	Base64 encoded

Are We Affected and Where? | SBOM

This is the one section of this reference guide that will likely require installation of additional tools. Below is a list that you may find useful in your investigation.

SBOM and Other Tools

cdxgen	https://github.com/CycloneDX/cdxgen
blint	https://github.com/owasp-dep-scan/blint
syft	https://github.com/anchore/syft
grype	https://github.com/anchore/grype
osv-scanner	https://github.com/google/osv-scanner
emba	https://github.com/e-m-b-a/emba
cyclonedx-cli	https://github.com/CycloneDX/cyclonedx-cli
sbomqs	https://github.com/interlynk-io/sbomqs
sbomgr	https://github.com/interlynk-io/sbomgr
sbomdiff	https://github.com/anthonyharrison/sbomdiff
Dependency Track (web app)	Web application designed to manage SBOMs, track dependencies, and monitor vulnerabilities
CycloneDX Tool Center	https://cyclonedx.org/tool-center/
SPDX Tools	https://spdx.dev/tools/

Creating SBOM in the middle of an incident is not ideal, but may be the best approach to get visibility into your exposure. The challenge is what to create SBOMs of, and what types of SBOM.

1. Identify affected products or components.
2. Generate SBOMs of critical software, or suspected affected software.
3. Monitor supply chain threat intel for new products affected.
4. Consider binary and firmware reversing to explore further.
5. Analyze SBOMs for vulnerable or malicious components.

SBOM Generation

syft scan <container> -o cyclonedx-json	Syft can also scan a project path, but is designed primarily to work with containers for SBOM generation
blint sbom -i <path> -o <file>	Bling analyzes binaries and generates CycloneDX SBOM from them.
./emba -f <firmware> -p ./scan-profiles/sbom-default.emba	Using emba and the sbom scan profile, emba runs many checks and is very slow, but using the sbom profile will eliminate some unnecessary tests
cdxgen -t <python/npm/maven> -o sbom.json <path>	

SBOM Inspection

cyclonedx-cli validate --input-file sbom.json	SBOM validator from CycloneDX
sbom-diff sbom1.json sbom2.json	SBOM comparison between 2 files to determine differences. Useful for comparing software versions or different tool outputs
sbomqs score sbom.json	Produce a quality score for the SBOM to determine how high quality it is
sbomgr packages -crEN "<package name>" <bom repo path>	Use sbom grep to recursively look for package names, commits, etc with a repository of SBOM files

Yara cli Syntax

yara rule.yar -r ./path	Recursively search path
vol.py -f memory.img yarascan --yara-file=rule.yar	Using yarascan plugin with volatility to scan memory dumps

What is the Impact?

Using what we know from prior phases—what is the worst thing that could happen to your organization due to this incident?

Impact								
Business					Technical			
Financial	Reputation	Non-Compliance	Privacy	Safety	Confidentiality	Integrity	Availability	Accountability

It's important to determine if your business functions have any requirements for these technical impacts before you get concerned with their loss.

Other factors include exploitability, ease of exploitation, reachability analysis and network exposure.

https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

Assess Vulnerabilities

curl -s "https://services.nvd.nist.gov/rest/json/cves/2.0?cveId=CVE-2021-44228"	Interrogate NVD for CVEs, this is the default source for many vulnerability tools
curl -s https://api.osv.dev/v1/vulns/<id>	Interrogate OSV for CVE, OSV, MAL and many other vulnerability IDs from osv.dev.
curl -s "https://api.first.org/data/v1/epss?cve= CVE-2021-44228"	Determine if the vulnerability has a high likelihood of exploitation
https://github.com/cisagov/vulnrichment	Determine other factors such as exploitation, automation and other impact criteria using CISA vulnrichment and SSVC

You can enhance these results with improved parsing, using other utilities:

```
curl -s "https://api.osv.dev/v1/vulns/CVE-2021-44228"
    | jq '.affected[] | {product: .package.name,
         versions: .versions}'
```

Vulnerability Exploitability eXchange (VEX) is a companion to SBOM that seeks to answer the question whether the software is truly affected by the vulnerability. It is not actually a statement of exploitability.

VEX – Not Affected				
Component_not_present	Vulnerable_code_not_present	Vulnerable_code_not_in_execute_path	Vulnerable_code_cannot_be_controlled_by_adversary	Inline_mitigations_already_exist

Other Tools for Vulnerability Response

https://secvisogram.github.io/	Edit and generate CSAF records (VEX and VDR)
https://vulnogram.github.io/	Edit and generate CVE advisories
https://github.com/aboutcode-org/vulnerablecode	Open source vuln database with 25+ importers
https://www.misp-project.org/	Malware threat sharing platform
https://github.com/volatilityfoundation/volatility3	Volatility memory extraction framework (the 2nd yara command uses this)
https://github.com/kevoreilly/CAPEv2	CAPEv2 malware sandbox
https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/	The ultimate list of SANS cheat sheets
https://www.sans.org/profiles/tony-turner/	Defending product supply chains authors and resources