Invary On-Premise Quick Start Guide

The Invary Runtime Integrity Platform measures and appraises the integrity of your operating systems in real time, finding hidden and zero day malware on your systems that are invisible to other threat detection solutions.

This guide provides a step-by-step set of instructions to quickly get started with Invary's on-premise solution for testing and evaluation. Full documentation is included with Invary's packages.

Step 1: Binary Overview

You will be installing the following Invary binaries:

1. **Invary Sensor**, which measures a virtual or physical machine to be appraised for its Runtime Integrity.
    a. Packaged as *invary-sensor.tar.gz*
2. **Invary Appraiser**, which appraises measurements provided by an Invary Sensor.
    a. Packaged as *invary-appraiser.tar.gz*
3. **Invary Baseline Library**, a set of files that are the baselines expected of an OS.
    a. Packaged as *invary-baselines.tar.gz*
4. Optional: Invary Baseliner, a binary capable of baselining an OS that has a custom kernel.
    a. Packaged as *invary-baseliner.tar.gz*
5. Optional: Sample Ansible Scripts
    a. Invary provides example Ansible scripts that automate this quick start guide.

Step 2: OS Support and Baseline Setup

Baseline once, measure anywhere

Invary Baselines are binary models of expected behavior of an OS, used by the Invary Appraiser to validate the Runtime Integrity of your running systems. A single Baseline is scoped to a distribution & kernel version, and can appraise any machine running that distro+kernel pair.

OS Support

Invary supports the following distributions out of the box: Alma Linux, AWSLinux, CentOS, Debian, Photon, Redhawk, RedHat, Rocky, and Ubuntu, with any kernel version that supports eBPF.

Steps to deploy baselines:

1. If you are using a COTS distribution:
    a. Extract the baselines to */var/opt/invary-baselines*
    b. Note you do not need to persist baselines for OSes you do not use.
2. If you use a custom kernel:
    a. Boot a virtual or physical machine with your custom kernel and modules.
    b. Ensure this is a trusted machine with no modifications other than your intended kernel updates and kernel modules used in production.
    c. Place the Invary Baseliner binary on the machine.
    d. **Run:** `sudo ./invary-baseliner --output $(uname -r).baseline`
        i. The resulting file should look something like this: `6.1.0-9-amd64.baseline`
        ii. **Place the resulting file in the appropriate OS folder in the baseline directory** for that distribution e.g.:
            1. /var/opt/invary-baselines/debian/12/x86_64.

Step 3: Install Invary Appraiser and Sensor

1. **Extract the Invary-Appraiser on the machine you want to run the appraiser on.** It can be a physical or virtual machine.
2. Run the invary-install.sh script:
    a. `sudo ./invary-install.sh -d`
3. Your Invary appraiser is now running and will output any appraisals to */var/opt/invary-appraiser/appraisals*
4. **Extract the Invary-Sensor package and place the files on the same machine**.
5. **Run**: `sudo ./invary-install.sh -d`

The Appraiser and Sensor are now running on the same machine with appraisal outputs being written to /var/opt/invary-appraiser/appraisals.

To learn how to configure additional options, including streaming appraisal results via a webhook, please see the full Invary On-Premise Guide or readmes contained with your Invary package.

Step 4: Testing

Invary has two open source projects to help you test a true positive scenario for failed Runtime Integrity.

1. Invary Test Probe: https://github.com/Invary-Runtime-Integrity/invary-test-probe
    a. The Invary Test Probe is a very simple Linux kernel module which inserts a symbol into the kernel symbol table for discovery by the Invary sensor software. It does not modify the running system in any other way.
2. Invary Test Kit: https://github.com/Invary-Runtime-Integrity/invary-test-kit
    a. The Invary Test Kit is a Linux kernel module which hijacks the kill system call, resulting in an error detected by the Invary sensor software. It does not modify the running system in any other way, and only logs to syslog that kill has been called before passing control back to the normal system call.

Once you have installed either kernel module the next appraisal will result in a failure. It is not recommended to leave either kernel module installed once you are done testing.

If you are interested in more comprehensive true positive testing please contact the Invary team at support@invary.com

Appendix A: Appraisal Output Guide

By default each endpoint is appraised once every 15 minutes (configurable). The resulting appraisal has the following JSON payload:

Appraisal

The top level appraisal object:

Field	Description	Type	Example
id	Unique id of the appraisal	id	`nkcxes7cwnd7dsyfwdhqc96ab`
created	Date and time when the appraisal was performed	date time	`2023-10-20T15:19:02.850349Z`
status	Overall status of the appraisal	One of: UNKNOWN, SUCCESSFUL, FAILED, ERRORED	`SUCCESSFUL`
endpoint	Unique id of the endpoint / host	id	`w6rq1heetdznah7btt7n5srw2`
passed	List of checks passed	List of check	`[`

		name	<pre>"required.nodes",
			"function.pointers",
			"task.gates",
			"task.tree",
			"data.nodes",
			"nops.table"
			]</pre>

failed	List of checks that failed. When the status is FAILED, this field highlights which check(s) failed.	List of check name string	<pre>[
			"jump.tables"
			]</pre>

measurement	Information about the measurement that the appraisal was based upon	**Measurement**	See below
kernel	Details about the endpoint's kernel	**Kernel**	See below
node	Details about the endpoint	**Node**	See below
distribution	Details about the endpoint's operating system	**Distribution**	See below
checks	Details about the checks run as part of the appraisal	list of **Check**	See below

Measurement

The measurement details object

Field	Description	Type	Example
nodes	Number of nodes in the measurement graph	number	`110847`
edges	Number of edges in the measurement graph	number	`378929`
modules	Number of kernel modules measured	number	`80`

Kernel

The appraisal/kernel details object:

Field	Description	Type	Example

name	Name of the operating system kernel	string	`Linux`
release	Kernel release string	string	`5.10.0-25-amd64`
version	Kernel release version	string	`#1 SMP Debian 5.10.191-1 (2023-08-16)`

Node

The appraisal/node endpoint details object

Field	Description	Type	Example
processor	Processor architecture string	string	`x86_64`
name	Hostname	string	`host1`
uptime	Uptime (time since last boot) in seconds	number	`538`
tags	Metadata tags assigned to this host via the sensor config file	List of string	`[ "asset_32703", "lab" ]`

Distribution

The appraisal/distribution operating system details object

Field	Description	Type	Example
vendor	OS vendor	string	`debian`
release	OS release	string	`11`
codename	OS release codename	string	`bullseye`

Check

The appraisal/check details object

Field	Description	Type	Example
name	The name of the appraisal check	string	`required.nodes`
result	Outcome of the check	One of:	`FAILED`

		UNKNOWN, SUCCESSFUL, FAILED, ERRORED	
failureDetails	Details of the check (when result is FAILED)	FailureDetails	See below

FailureDetails

The appraisal/check/failure details object

Field	Description	Type	Example
atNode	Location of the failure in the measurement graph in the form of: *symbol_name* or *<module_name> memory_address @symbol_name*	string	`<invary-core-kernel-module:text> @ffffffffae2002e0:sys_call_table`
reason	A human-readable explanation for the failure	string	`Matching sys_call_table entry could not be found in baseline`
value	Values associated with the failure	List of string	`Missing kernel__x64_sys_read, found rogue:840 instead.`

Successful Appraisal Example

When an endpoint's operating system has integrity, the appraisal will be successful:


```
{
  "id":"paaht7ec1scrd0g49p3w4fxkf",
  "created":"2024-01-26T21:58:03.180105108+00:00",
  "status":"SUCCESS",
  "distribution":{
    "codename":"bullseye",
    "release":"11",
    "vendor":"debian"
```



```
  },
  "endpoint":"s24fc74bk397cr3p7x605vn2f",
  "kernel":{
    "name":"Linux",
    "release":"5.10.0-26-amd64",
    "version":"#1 SMP Debian 5.10.197-1 (2023-09-29)"
  },
  "measurement":{
    "edges":"53508",
    "modules":"76",
    "nodes":"10997"
  },
  "node":{
    "name":"host1",
    "processor":"x86_64",
    "uptime":"6567",
    "tags":["tag1","tag2"],
  },
  "passed":["required.nodes", "task.tree", "task.gates",
            "jump.tables", "data.nodes", "nops.table",
            "function.pointers"],
  "failed":[],
  "checks":[
    {"name":"required.nodes", "result":"SUCCESSFUL"},
    {"name":"task.tree", "result":"SUCCESSFUL"},
    {"name":"task.gates", "result":"SUCCESSFUL"},
    {"name":"jump.tables", "result":"SUCCESSFUL"},
    {"name":"data.nodes", "result":"SUCCESSFUL"},
    {"name":"nops.table", "result":"SUCCESSFUL"},
    {"name":"function.pointers", "result":"SUCCESSFUL"},
  ]
}
```


Failed Appraisal Example

This example of a Failed Appraisal was created by using the Invary Test Kit, with pertinent content in bold:


```
{
  "id":"paaht7ec1scrd0g49p3w4fxkf",
  "created":"2024-01-26T21:58:03.180105108+00:00",
  "status":"FAILED",
  "distribution":{
    "codename":"bullseye",
    "release":"11",
    "vendor":"debian"
  },
  "endpoint":"s24fc74bk397cr3p7x605vn2f",
  "kernel":{
    "name":"Linux",
    "release":"5.10.0-26-amd64",
    "version":"#1 SMP Debian 5.10.197-1 (2023-09-29)"
  },
  "measurement":{
    "edges":"53508",
    "modules":"76",
    "nodes":"10997"
  },
  "node":{
    "name":"host1",
    "processor":"x86_64",
    "uptime":"6567",
    "tags":["tag1","tag2"],
  },
  "passed":["required.nodes", "task.tree", "task.gates",
            "data.nodes", "nops.table", "function.pointers"],
  "failed":["jump.tables"],
  "checks":[
```



```
    {"name":"required.nodes", "result":"SUCCESSFUL"},
    {"name":"task.tree", "result":"SUCCESSFUL"},
    {"name":"task.gates", "result":"SUCCESSFUL"},
    {"name":"data.nodes", "result":"SUCCESSFUL"},
    {"name":"nops.table", "result":"SUCCESSFUL"},
    {"name":"function.pointers", "result":"SUCCESSFUL"},
    {"name":"jump.tables", "result":"FAILED"
     "failureDetails":[ {
        "atNode":
          "<invary-core-kernel-module:text>
          @ffffffffae2002e0:sys_call_table",
        "reason":
          "Matching sys_call_table could not be found in baseline",
         "value":[
           "Missing <invary-core-kernel-module:text>
           @ffffffffa90a4230:__x64_sys_kill,
           Found <invary_test_kit:text>
           @ffffffffc0a5b346:invary_kill instead"
         ]
      } ]
    }
  ]
}
```

This failed result contains a few key pieces of information:

- It shows what segment of the kernel has been impacted, in this case the System Call section of the Jump Tables category. Other categories include: Required Nodes, Task Tree, Task Gates, Data Nodes, Nops Table, and Function Pointers.
- It shows it was unable to find the expected *kill* system call and instead found *invary_kill* in its place; showing a substantive change in definition.
- It is showing the memory address of the expected *kill* system call along with the memory location of the *invary_kill* call that replaced it; showing a substantive change in expected memory location.

Appraisal results also contain helpful information, like any tags you may have supplied this particular machine to help identify it and what OS and Kernel it is running.

Integration

Appraisal events are JSON formatted for easy ingestion into your SIEM, CNAPP, XDR, or log aggregation system.

The quick start install outputs appraisals to disk in json format, while also persisting them in syslog. You can also configure a webhook to stream these events from the Invary Appraiser to your system. Invary has an example Opensearch install with predefined queries that showcases such ingestion via a webhook which is available upon request.

