GRR Artifacts

Greg Castle
Blackhat 2014
Contact: grr-dev@googlegroups.com

Abstract

During a security investigation responders need to quickly retrieve common pieces of information that include items such as logs, configured services, cron jobs, patch state, user accounts, and much more. These pieces of information are known as forensic artifacts, and their location and format vary drastically across systems. We have built a framework to describe forensic artifacts that allows them to be collected and customised quickly using the GRR Rapid Response open source live forensics system. We aim to provide a centralized, free, community sourced, knowledge base of forensic artifacts that the world can use both as an information source and within other tools.

Background

In a windows response scenario you might be interested in retrieving items such as the registry hives, event logs, user accounts, runkeys, running services, and ‘at’ jobs. Many investigators hold the location of all this information in their heads, internal wiki’s, text files etc. that have been built from years of experience, but this doesn’t scale well.

Increasingly investigators need to operate outside of their comfort zones on unfamiliar operating systems with unfamiliar software. In these scenarios investigators will rely on information such as forensicswiki, forensicartifacts.com, blogposts, conference slides and whatever other information is available. The problems here are:

- The information isn’t machine readable
- Accuracy and completeness varies wildly
- Information is often published once and never updated, and where updates are needed it’s often easier to republish than edit the old content
- There’s no consistent format to specify machine specific data like usernames, Windows user SIDs, home directories etc. that is needed to fully specify many important user-specific artifacts
- There are no good ways to build logical groups of information like “All persistence mechanisms”

Goals

The goals of the GRR artifacts implementation are:

- Describe artifacts with enough precision that they can be collected automatically without user input.
- Cover modern versions of Mac, Windows, and Linux and common software products of interest for forensics.
- Provide a standard variable interpolation scheme that allows artifacts to simply specify concepts like "all user home directories", %TEMP%, %SYSTEMROOT% etc.
- Allow grouping across operating systems and products e.g. “Chrome Webhistory” artifact knows where the web history is for Chrome on Mac/Win/Linux.
- Allow grouping of artifacts into high level concepts like “Persistence Mechanisms”, and investigation specific meta-artifacts.
- To create simple, shareable, non-grr-specific human-readable definitions that allow people unfamiliar with the system to create new artifacts. i.e. not XML or a domain specific language.
- The ability to write new artifacts, upload them to GRR and be able to collect them immediately.

Artifacts vs. IOCs

While there are some similarities with Indicators of Compromise (IOCs), the intent is quite different and artifacts are not intended to be IOCs on their own. An IOC might read something like this:

```
If filename “temp.exe” contains string “evil” or is signed by “stolen
cert”
```

In contrast a GRR artifact definition would be purely data, no logic, and describe legitimate system state rather than malware state like, the user runkeys are at:

```
HKEY_USERS\%%users.sid%%\Software\Microsoft\Windows\CurrentVersion\Ru
n\*
```

There are many, many standards in this general space: OpenIOC, CyBox, STIX, MAEC, CAPEC, TAXII, Oval, SCAP, IODEF, Yara, Veris, IDMEF, but none tackle the artifact problem as described above.

GRR Artifacts

GRR artifacts are defined in YAML, with a style guide here. We use a standard set of machine information collected from the host for variable interpolation. This collection of data is called the knowledgebase and is referenced with a %%variable%% syntax. A simple artifact example is:

```
name: WinPathEnvironmentVariable
doc: The %PATH% environment variable.
collectors:
- collector_type: REGISTRY_VALUE
  args: {path: '%%current_control_set%%\Control\Session
Manager\Environment\Path'}
provides: [environ_path]
supported_os: [Windows]
urls: ['http://environmentvariables.org/WinDir']
```

Dependencies (e.g. this artifact depends on the current_control_set from the knowledgebase) are determined by analysing paths and finding corresponding “provides” entries.

Differences across OSes are handled simply:

```
name: SafariHistory
doc: Safari browser history (History.plist).
collectors:
- collector_type: FILE
  args:
    path_list:
      - '%%users.localappdata%%\Apple Computer\Safari\History.plist'
      - '%%users.appdata%%\Roaming\Apple
Computer\Safari\History.plist'
  supported_os: [Windows]
- collector_type: FILE
  args:
    path_list:
      - '%%users.homedir%%/Library/Safari/History.plist'
  supported_os: [Darwin]
labels: [Browser]
urls: ['http://www.forensicswiki.org/wiki/Apple_Safari']
```

as are differences between OS versions:

```
name: SetupApiLogs
```

```
doc: Windows setup API logs.
collectors:
- collector_type: FILE
  args: {path_list: ['%%environ_systemroot%%\setupapi.log']}
  conditions: [os_major_version < 6]
- collector_type: FILE
  args:
    path_list:
      - '%%environ_systemroot%%\inf\setupapi.app.log'
      - '%%environ_systemroot%%\inf\setupapi.dev.log'
      - '%%environ_systemroot%%\inf\setupapi.offline.log'
  conditions: [os_major_version >= 6]
labels: [Logs]
supported_os: [Windows]
urls: ['http://www.forensicswiki.org/wiki/Setup_API_Logs']
```

The artifact defines where the data lives. Once it is retrieved by GRR a parser can optionally be applied to turn the collected information into a more useful format, such as parsing a browser history file to produce URLs.

To re-use GRR artifacts a project needs to:

1. Support path globbing: the `\file\path\*\something` syntax is used extensively.
2. Have a means to populate some or all of the knowledgebase information, such as from a live system, dead disk analysis, or memory analysis. Most artifacts can be supported with a minimal set of information (usernames and homedirs on OS X and Linux, %SystemRoot%, CurrentControlSet and some user data on Windows).
3. Be able to ingest YAML (support exists for by most programming languages)

GRR currently defines around 130 different artifacts and 30 parsers. Artifacts have already proven very useful in real-world investigations.

Future Work

The format will likely change slightly in the near future as we iterate on the GRR artifact collection implementation, but any conversion required from the existing artifact should be simple. Inside GRR we want to make collecting large numbers of artifacts even faster, make dependencies more obvious, and debugging an artifact dependency chain easier. We're also thinking about using strong typing with artifacts to enable sharing and re-use of parsers, and simpler processing of results outside of GRR.

The Plaso project (formerly log2timeline) developers are actively investigating using GRR artifacts and we welcome other collaborators. The Rekall framework may also re-use some artifacts.

We have experimented with using artifacts for complex fleet checking, sort of an IOC minus the logic. This approach combined with a logic engine like Yara, could provide a full-fledged IOC capability in the future.