Linux Built-in Security (SELinux)

SELinux is a security feature on most Linux systems to allow more granular access controls to files and processes. SELinux uses security labels or security context to classify resources such as files and processes. This context allows SELinux to enforce rules as to who is allowed access to a resource and how they are allowed to access it. The security context is stored as an extended file attribute and is associated with the files inode, so that the security context will remain no matter if the file was moved or renamed. The security context consists of four fields; User, Role, Type and Level.

USER: The first component of the security context is the SELinux User component. This component can be thought of as a way of grouping roles.

SELinux Users can have multiple roles that they can reach, and then in those roles they can reach multiple types. Three users that you will usually see on the system are `user_u`, `system_u` and `root`.

- `user_u` is the default SELinux User for a logged in user on a system
- `system_u` is the default User for processes started during the boot up process, i.e., they were never started by a user
- `root` is the SELinux User that you get when you login at the console as "`root`"

In targeted policy, the user component is really not important. It is more important in MLS and Strict policy machines. On a file, the user component specifies the SELinux User that created the file. The initial system files are labeled as `system_u`, or if you relabel, they will get set back to `system_u`. By convention all default SELinux Users end with a `_u` except for root. Originally if you wanted to map a Linux User to an SELinux User, you would create a SELinux user name with the same name as the Linux User. This is why the root user does not end in a `_u`.

ROLE: The second component of the security context is the Role field. This field is only really relevant on processes or domains. The role field on a file is always `object_r`, and really has no meaning other than as a place holder. You would usually see a role like `system_r` or `sysadm_r` on a process. Roles are used to group security types so you can specify (in policy) which roles are able to execute which types. This is the basis of Roles Based Access Control (RBAC) in SELinux. RBAC is not really used in targeted policy, but becomes more important in Strict and MLS policy. MLS Policy also contains `sysadm_r`, `staff_r`, and `secadm_r`. Roles by convention end with a `_r`.

TYPE: The 3rd component of the security context is the Type component, for example /usr/sbin/httpd is labeled with a type of `httpd_exec_t`. This is the heart of SELinux Type Enforcement. Most of the policy rules in SELinux revolve around what subject types have what access to which object types. By convention this component always ends in a `_t`.

LEVEL: On strict and targeted policies, we refer to this as the MCS (Multi Category System) Field. Unfortunately, this field can contain a ":". The syntax of this field can look something like `s0-s15:c1,c2` but syntax will be discussed at a later date. Most files by default are labeled `s0`, sometimes referred to as SystemLow. Fortunately SELinux provides a translation library (libsetrans) that replaces the code in this field with a more human readable form. For example, something like `s0:c1,c2` might show up to the user as `PatientRecord,CompanyConfidential`. On targeted or strict policy machines, `s0` translates to "", which means that all files will not show the fourth field.

Though not officially built in by default, due to the enhancements enabled by this program, we want to include Grsecurity in our discussion of Linux Security capabilities. GRSecurity is an extensive security enhancement to the Linux kernel that defends against a wide range of security threats through intelligent access control, memory corruption-based exploit prevention, and a host of other system hardening that generally require no configuration. These enhancements are very simple to configure since they are basically "enabled out of the box". Unfortunately, to get these enhancements the user must custom compile the kernel with the grsecurity patch unless your particular distribution offers this package from their repository. Once it is compiled into the kernel, runtime configuration is controlled through the `sysctl` command.

GRSECURITY provides several security enhancements for Linux. These include:

RBAC: RBAC is intended to restrict access to the system further than what is normally provided by Unix access control lists, with the aim of creating a fully least-privilege system, where users and processes have the absolute minimum privileges to work correctly and nothing more. This way, if the system is compromised, the ability of the attacker to damage or gain sensitive information on the system can be drastically reduced.

PaX: is a patch for the Linux kernel that implements least-privilege protections for memory pages. PaX flags data memory as non-executable, program memory as non-writable and randomly arranges the program memory. This effectively prevents many security exploits, such as some kinds of buffer overflows. The former prevents direct code execution absolutely, while the latter makes return-to-libc (ret2libc) attacks difficult to exploit, relying on luck to succeed. PaX does not prevent variable and pointer overwriting.

CHROOT Restrictions: grsecurity restricts chroot in a number of ways to prevent some vulnerabilities and also privilege escalation attacks. Some chroot modifications include:

- No attaching shared memory outside chroot,
- No getsid outside of chroot,
- No viewing of processes outside of chroot
- No mounting or remounting

ADDITIONAL AUDITING: grsecurity adds additional logging and auditing. Items such as kernel changes, processes execution, execution inside a chroot jail, ptrace actions, directory changes and many more can be captured.

TRUSTED PATH EXECUTION: Trusted path execution is another optional feature that can be used to prevent users from executing binaries that are not owned by the root user, or are world-writable. This is useful to prevent users from executing their own malicious binaries or accidentally executing world-writable system binaries that could have been modified by a malicious user.

Signature and Heuristic-based Detection

Signature detection engines use a pre-compiled database of known bad malicious code. This is similar to IDS systems using signatures of known malicious traffic for their alert engines.

Signatures can contain all kinds of different elements. File sizes, keying in on certain bytes in the code, changes to a particular registry key, or calling a specific memory address for example.

The drawback to this detection method is that the malware needs to be known by the particular A/V vendor in order to be in the database. Unknown or 0-Day code will go undetected by this type of engine.

- Uses a pre-compiled database of signatures for known malicious code
- A signature is combination of elements
    - Bytes in file + Strings in memory + File hash = **Hit!**
    - Malware must be "caught" and analyzed first
- Normally, updates to the signature database must come from the ESP vendor
- Usually does not produce false positives but occasionally does

Signature-based Detection

Pros	Cons
Rarely wrong	Reactive detection
Quick	Bypassed by altering binary or using packers (although this could be a signature itself, i.e., UPX)
Can provide full identification of malicious element	Signatures vary by region/subsets of users
Often able to repair or restore	Adding more signatures makes the ESP slower
Use few CPU/Memory resources	

Using signature detection engines have both good points and bad points. On the good side, they are very quick and accurate. The A/V can usually provide a fair amount of detail about the threat and can usually fix or repair the problem. On the downside, signature databases, by their nature, can't detect the unknown. The time from discovery of some piece of code to the publishing of an effective signature varies.

Since detection relies on known specific things, morphing the code or packing it can allow the malware to again bypass detection.

As mentioned, signature detection is usually very accurate however occasionally they will flag something that is not malware, such as a windows patch or some legitimate 3rd party program.

Signature Creation

It usually takes 2-4 weeks to analyze malware, write signatures, test on variants, run through QA and distribute.

Not all malware is treated equally by A/V vendors. Much of the malware they receive doesn't even warrant a signature. It's a time versus money trade off. If there is a chance of a large scale infection of their customer base, then they will create a signature. If not, then they aren't going to worry about it.

If the vendor does decide to create a signature, it normally takes between 15-30 days for the signature to be published. This doesn't include malware that hits the public media's attention. If pushed, A/V vendors can push out a signature in hours.

Not all vendors automatically get all malware right away, however A/V vendors do share their malware and findings. Usually a week after publishing their initial signature, the vendor will share the code with the rest of the community. Again, other vendors, if they do put out a signature, will take between 15-30 days unless pushed to put the signature out.

Heuristic-based Detection

BEHAVIORAL ANALYSIS:

- Watch for OS function calls (LoadLibrary, SetWindowHook, etc).
- Analog scale: Is the process listening? Modifying the registry? Modifying AppInit?
- VM analysis runs for a few seconds.

- Heuristic detection engines don't rely on known signatures to detect potentially malicious activity, they look instead at the activity occurring or the code base of the program, in effect making a best-guess as to whether something is malicious or not.
- An example of heuristic detection would be detecting a program that was packed with a known packer and flagging the process for that reason alone. Another example would be flagging a process that writes a value to the AppInit key in the registry.
- Either of these activities could be totally legitimate, but since they are also known to be used by malware, they will get flagged as suspicious.
- Many times, if an A/V product has a Heuristic engine, it will either be off by default or turned to it's lowest level to cut down on false positives.
- Uses behavioral analysis to identify malware.
    - Run the suspect executable in a mini-virtual machine and monitor for malicious behavior.
    - Compares behavior with that of known malware.
- Does not require signatures or advance knowledge of the specific malicious code.
- Essentially, the ESP making an educated guess that something is malicious.

Pros	Cons
Can detect unknown variants of known malware	Higher degree of false positives
Simple changes in code do not always prevent detection • Polymorphic malware	Slower execution and detection
	Not easily able to repair
	Defeated by novel tools and techniques
	More CPU/Memory Intensive

Just like anything else, there are pros and cons to heuristic detection engines. They are very good at discovering or detecting unknown malware, but they are more labor intensive on the machine and are apt to throw false positives. This final con is one of the main reasons that many users either turn off this capability or limit it to such an extent as to be useless.

ESPs – Common Questions and Answers

As an operator, there are some questions you will need to ask yourself when it comes to ESP concerns. Here are some common questions and answers.

Question	Answer
Is the A/V regularly updated?	For example, free A/V or commercial A/V indicates that the organization has limited resources Enterprise A/V indicates that security is a concern. Centralized updates and logging indicates that security is a priority.
Does the ESP protect the system in real-time?	If not, the system activities could be caught at any time.
Is there a scheduled time to run a system scan?	Suspicious activities may not be caught by real-time protection but a scheduled system scan runs deeper. Perhaps the system scan takes so much processing power that the system will stop responding to your commands during that time.
Will the ESP catch my tools and if so what actions will the ESP take?	It could quarantine the tools and place them in a directory.
If so, where is that directory located?	Some ESPs will forward malware to their lab.
What are the settings of the ESP?	Many ESPs use XML-based configs or have the settings in the registry. Look at the settings to see if they are set to the default or are modified.

Where are the logs and are your actions in the logs?	Logs may be saved locally or shipped off to some central logging facility.
Does this ESP send pop-ups and alerts to the user's desktop?	Find out if the user is logged in or is the screen saver on. You may have caused a pop-up to appear.
Is this ESP a threat?	Research if the ESP is a threat by looking at the following: • Registry Key Locations vs. Log Locations • Real-Time capability vs. Scheduled scanning • Program Settings (Default vs. modified) • Capture vs. Quarantine vs. Forwarding of your tools • Logging of your actions • Pop-ups and Alerts

Network Monitoring Software

Multi Router Traffic Grapher (MRTG): Is one of the most popular implementations for monitoring network activity. MRTG monitors and graphs SNMP network devices. MRTG is written in perl and works on Unix/Linux as well as Windows and even Netware systems.

ntop: Is a network traffic probe that shows network usage. It can run on Unix or Windows systems. The management interface is accessed through a web browser and listens on port TCP/3000 by default. ntop has the ability to:

- Sort network traffic according to many protocols
- Show network traffic sorted according to various criteria
- Display traffic statistics
- Store on disk persistent traffic statistics in Round Robin Database (RRD) format
- Identify computer users, without sending probe packets
- Identify the host OS
- Show IP traffic distribution among the various protocols
- Analyze IP traffic and sort it according to the source and destination
- Display IP Traffic Subnet matrix that is, who's talking to who
- Report IP protocol usage sorted by protocol type
- Act as a netflow or sflow collector for flows generated by routers, such as Cisco and Juniper or switches, such as Foundry Networks and Produce RMON-like network traffic statistics. Something like this could easily spot activity on non-standard ports of a given IP.

IDS: An intrusion detection system (IDS) is software and/or hardware designed to detect unwanted attempts at accessing, manipulating, and/or disabling computer systems.

IDS can be composed of several components:

- Sensors generate security events,
- Consoles monitor events and alerts and control the sensors, and
- Central engines record events logged by the sensors in a database and uses a system of rules to generate alerts from security events received.

There are several ways to categorize IDS depending on the type and location of the sensors and the methodology used by the engine to generate alerts.

Recommended Internet Sites

- Well known “false positive” examples: MARCH 2006: McAfee vs. Excel: https://web.archive.org/web/20160809182446/http://www.cnet.com/news/mcafee-update-exterminates-excel/
- APRIL 2010: McAfee vs. svchost (endless reboots): https://web.archive.org/web/20160804111417/http://krebsonsecurity.com/2010/04/mcafee-false-detection-locks-up-windows-xp/
- To learn more about MRTG, go to their Web site at: https://web.archive.org/web/20160804111548/http://oss.oetiker.ch/mrtg/
- To learn more about ntop, go to their Web site at: https://web.archive.org/web/20160922150506/http://www.ntop.org/

Please contact the Course Coordinators if you are unable to access any of the Recommended Internet Sites.