BUILDING A MALWARE LAB IN THE AGE OF BIG DATA

Vanja Svajcer
HP, Croatia

Email vanja.svajcer@hp.com

ABSTRACT

The majority of established industry malware research labs have been built organically over the course of years – even, in the case of many founding members of the anti-virus field, over decades.

A lab usually started as a simple virus repository – a physically isolated space in which samples were stored in a secure fashion so that they could only be accessed, analysed and tested by skilled and authorized researchers. Over time, sample-collection systems such as honeypots, spam traps and web crawlers were added, along with systems for sharing samples between trusted vendors.

Eventually, these systems, together with the increase in activity by malware writers, raised sample volumes to a point where human researchers could not process all samples manually. This point marked the start of the age of automated analysis.

Soon, though, existing automated analysis systems were inundated with ever-increasing traffic volumes. The need for clustering, correlation and automated classification became clear. All this organic growth caused malware labs to become extremely complex, with systems that were interdependent and tied to the existing technology used by each company's products.

Recently, we have seen an increase in the number of newcomers to the field of malware research, who each bring their own ideas as to how malware problems should be tackled. These newcomers include incident response companies as well as the emergency response teams of large companies and government organizations, and they all need their own labs.

Unfortunately, it is not always clear how to successfully evaluate available options and start building an integrated environment for threat collection, analysis, correlation, and incident tracking and management. There is a clear need for a process that can be followed to build a malware research lab from scratch.

Our paper will propose a simple process for building a fully functional malware research lab in a relatively short time. It will provide criteria for evaluating existing systems in each of the mandatory areas of a fully functional malware lab: collection, analysis, classification, protection, testing, sharing and integration.

INTRODUCTION

Most experienced malware researchers started their careers in an established anti-malware laboratory owned by one of the anti-virus companies. These are often referred to as 'virus labs' or simply 'labs' (two terms used interchangeably through the rest of this paper). A virus lab's processes and procedures are governed by a strict set of rules, which prescribe secure and safe practices for handling potentially malicious samples, from their import into the secure environment to their processing and storage.

It has been common practice to keep virus labs physically separated from the rest of their companies' production systems, with physical access granted only to researchers and a small group of trusted individuals. This approach guaranteed that none of the malicious files ever left the safe confines of the lab before being rendered unusable for the external environment. This was usually accomplished by encrypting samples with PGP, which remained one of the best industry sample handling practices until today.

Working in an anti-malware vendor's virus lab entails following additional rules to ensure the best possible service for users of anti-malware products. Virus labs are on the critical path to delivering timely and effective protection against malware attacks. With growing volumes of malware and increased user requirements it is not accidental that there are rigorous controls at every stage of the malware processing cycle, all the way from the collection of malware samples to the delivery of malware protection data.

Today, however, a growing number of smaller teams require virus labs and systems for manual and automated malware processing. The requirements of those teams are somewhat different from the requirements of the labs used by established anti-malware vendors. These differences stem primarily from the fact that these smaller entities do not have to collect, process and store every single malware sample, and are not tied to any particular technology or protection technique. Examples of such entities include CERT and CIRT teams, threat intelligence teams, malware SOC analysts, malware research and response teams in large companies, and complementary detection technology teams.

This paper provides an introduction to virus lab components, design, and development principles for teams that have a clear need for implementing a virus lab and do not yet have a fully functional lab. For readers with high-throughput virus labs, such as ones owned by established anti-malware vendors, the paper may also help as a reference to compare their own facilities with the proposed design and development guidelines.

The emphasis of the paper is on describing components and development principles driven by the so-called new style of IT, which emphasizes the mobility of users, data and computing resources. With an ever-increasing volume of malware samples and malware metadata, and shrinking budgets available to malware research teams processing the latest threats, aspiring lab builders have to tread a narrow path to achieve optimum functionality given their time and resource constraints.

The paper concludes with recommendations and predictions for the future of virus labs and a catalogue of free, open source, and selected commercial components worth considering when building a virus lab. Many of these are used as *de facto* standards by the malware research community.

MALWARE LAB DESIGN: HISTORY

The first dedicated virus labs were built [1] in the early 1990s as an answer to increasing requirements for safe and secure storage of virus samples. There were only a handful of known viruses then, and the discovery of new strains was an infrequent occasion. Computer viruses of the early nineties were written for DOS and had no network functionality, which meant that the main requirements for a virus lab were physical separation and secure storage of media containing virus samples.

In addition to physical isolation and controlled access, there was a need for separation between systems for static and dynamic analysis. Each researcher usually had two physical machines – one used for static analysis and signature development and the other for replication and debugging. The shared virus lab's systems were used only for secure storage of samples.

The growth in number of newly discovered virus samples in the mid-nineties, although tiny by today's standard (just over 4,000 virus variants were known in 1995 [2]), was a major driver for the adoption of new analysis techniques. In 1993, researchers from *IBM* [3] first drew parallels between a human immune system infected by a biological disease and a computer system infected by a computer virus, which prompted work in the areas of computer immunization and development of automated analysis systems.

The authors of the seminal *Digital Immune System* (*DIS*) [4] devised a fully automated system, which was put into use towards the end of the decade. *DIS* was a result of truly ground breaking work. It had significant consequences both for the future development of virus labs as well as for systems built around the virus lab's core deliverables.

By the early years of the first decade of the 21st century, it became clear that no functioning virus lab would be able to fulfil its main task of processing all newly discovered malicious samples without the help of automated analysis systems. Building an automated analysis system became a strategic goal for any virus lab, and this is when the first generation of automated analysis systems saw widespread adoption.

Around the same time, desktop system virtualization products started maturing. These left a deep mark on the development of lab systems as well as on the overall computing and networking model in most medium to large enterprises. Today, it is impossible to think about manual malware analysis, let alone automated malware analysis, without the incorporation of virtualization and system virtual machines.

Soon after the first generation of automated analysis systems were built, usually based on physical machines and automated re-imaging processes, it was clear that those systems would not scale with the volume of malware without some heavy investments in hardware. The advantages of virtualization were obvious, with its horizontal scaling capabilities, seamless upgrade paths, powerful snapshotting, and development SDKs. Together with the increase in scalability and the ability to process a large volume of samples, the industry generated an abundance of malware metadata that lent itself to clustering and other machine-learning algorithms, which further drove requirements for RAM and CPU cycles and storage capacity.

The computing requirements of a fully functional lab started to grow in many directions, the most significant of those being:

- Machine learning, clustering and classification
- Data analytics and search
- Metadata database systems
- Sample acquisition and sharing systems
- System integration and workflow systems

However, with a constant need to process all incoming samples (a process which often feels like fire-fighting), lab systems have grown organically rather than systematically, with strong connections to the underlying proprietary protection technology.

The end of the first and beginning of the second decade of the 21st century brought a myriad of complementary protection, detection and remediation techniques to the anti-malware industry, successful largely due to the perception of failure of technologies used by traditional anti-virus vendors.

Many medium and large enterprises became concerned with targeted attacks conducted by organized criminal groups and other sophisticated actors. The term Advanced Persistent Threat (APT), carrying a different meaning depending on the geography of the target and the company that addresses attacks, was first introduced in 2006 [5] and has since served as a bogeyman used by marketing departments. However, the term was later adopted by security officers in many companies – despite being laughed off as marketing gobbledygook by many malware researchers working for traditional anti-virus companies.

Large enterprises usually have their own security teams working in Security Operation Centres (SOCs), concerned with protecting their own estates. Since malicious software is used as the main delivery method in the majority of successful attacks, it did not take long for security analysts to specialize in malware analysis, which also meant that having their own virus labs became a major requirement. This paper attempts to help those teams when making decisions and building their own virus labs to address the requirements of their organizations.

WHAT KIND OF LAB TO BUILD?

Virus lab requirements

The requirements of a virus lab largely depend on the role of the team that owns it. Only specialized vendors with hundreds of researchers and software engineers will have a need for a high-throughput lab designed to handle hundreds of thousands of samples on a daily basis. Most internal malware research teams will not need to manually process more than a handful of samples daily and up to a few thousand unique suspicious files using the automated analysis systems, depending on their need and ability to acquire new samples.

The required throughput, the skill of the analyst, and the type of output will largely drive requirements in the following areas:

- Storage capacity
- Networking bandwidth
- CPU capacity
- Logical components of the virus lab
- Privacy and security of samples

A complete high-throughput virus lab

Judging by the latest data obtained from *AV-TEST.org* [6], it is clear that the growing volume of newly discovered

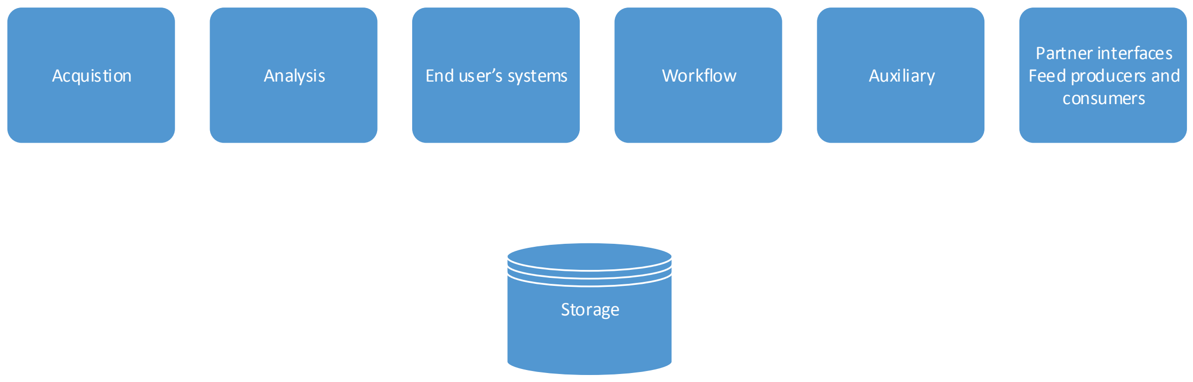

Figure 1: High-level functional units of a malware research and processing lab.

malware samples requires large capacities in all of the aforementioned categories for any lab wishing to process all known samples. The number of daily discovered samples has been growing and reached over 400,000 in 2014 [6]. In 2015, this number has stabilized, but it is difficult to tell whether the reason is stagnation in development of new malware or the industry's lack of capacity to discover even larger numbers of new samples.

Assuming over 400,000 daily samples with an average sample size of 500KB (and growing), the newly discovered samples require a daily storage capacity of 200GB (linearly extrapolated to 73TB/year if all samples are preserved) and network bandwidth of at least 20Mbps. That is assuming the uniform distribution of incoming malware samples throughout the day and no reserve capacity for when the number of daily discovered samples exceeds the average size. The peak bandwidth may well be in the range of hundreds of megabits per second. And that is only for handling the incoming samples, never mind all the other requirements of a high-throughput lab, which include sample sharing, data publishing and collection of telemetry.

High-throughput virus labs have grown organically over the course of many years, and designing one to be built in a timeframe of just a few months is out of the scope of this paper. We will concentrate in this paper on the lab requirements for smaller teams.

Optimal lab for smaller teams

To begin our design process, we must define the requirements and necessary components essential for a modern virus lab.

We will assume the throughput of the lab. No matter how small the throughput seems to be for anti-malware vendors, the virus lab we will design will not have an automated throughput higher than 10,000 samples a day at the beginning.

We will assume a team size of five researchers, which means they will be able to analyse manually in detail only a handful of malware samples every week.

We will define detailed analysis as the process of automated and manual analysis, which strives to extract all potential indicators of compromise with the intention of hunting the threat and its new variants in the organization's network. This may include deciphering the malware configuration and finding and extracting potential domain generation algorithms (DGAs) to prevent malware communication using egress filtering. We will also assume that the team will be able to create detection data intended to identify malicious samples for internal use before anti-virus protection signatures are in place.

In some cases, when the researchers are dealing with an attack on a high-profile target, anti-virus signatures will never be deployed, as the details of the attack tools, techniques and procedures (TTPs) will be known only internally (that is, inside the target) and not shared with any other entity; this is why the ability to create, test and deploy internal signatures is required.

HIGH-LEVEL LOGICAL DESIGN

Let us start our design with a very high-level overview of various classes of systems and then drill down into every functional group, its purpose, and its subcomponents. Figure 1 shows the high-level functional units of a malware research and processing lab.

Acquisition

If we think of a virus lab as a black box, the inputs required for any type of activity are malware samples. This remains true whether we are only looking into research on individual malware-related incidents, or undertaking classification and clustering of large malware sets consisting of thousands of malicious samples a day. The functional unit looking for and collecting samples is commonly known as Acquisition.

It is a fundamental imperative to be able to acquire malware samples as soon as they have been spotted in the wild. Reaction time is critical when dealing with incidents. Newly discovered malware could have been used by the attacker to get a foothold inside the target's environment, and it is important to analyse it quickly to gain knowledge and decide on the appropriate course of action when addressing the attack.

Let us consider the individual components of lab sample acquisition systems, as shown in Figure 2.

Building trust and participating in industry cooperation largely consists of consuming and producing various feeds including malware sample sets (a.k.a. collections) as well as metadata, domains, URLs, and other indicators of compromise (IOCs). This is why it is important to maintain systems that both consume samples and organize and publish sample sets to be consumed by sharing partners.

However, one should not only rely solely on third-party sources of data. Whilst a lot of cooperation is based on sharing, re-sharing samples is somewhat ethically

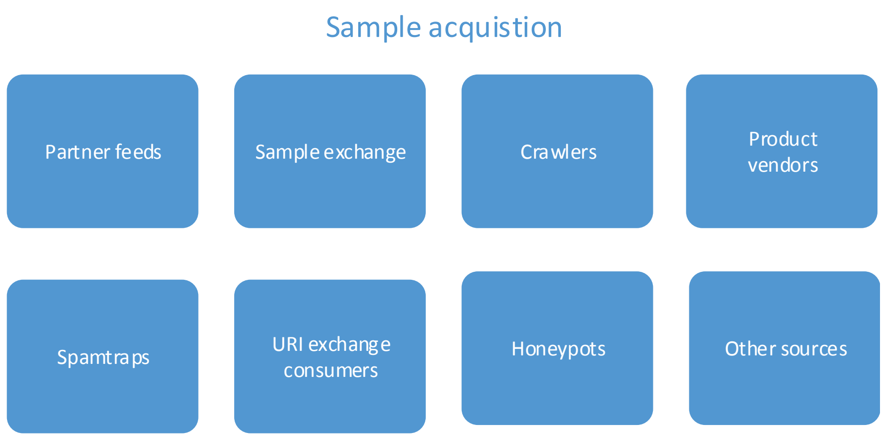

Figure 2: Sample acquisition components.

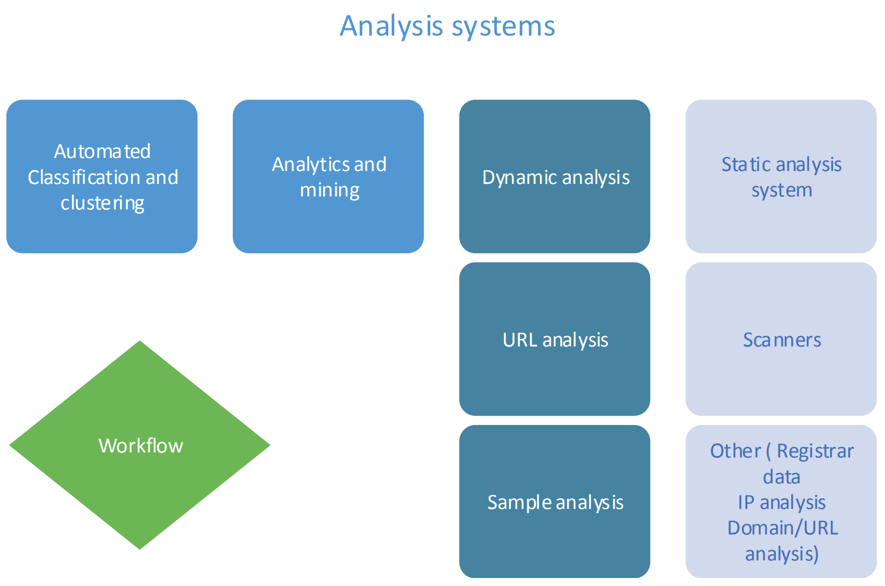

Figure 3: Components of internal automated analysis systems.

unacceptable. A fully functional lab has to be able to collect its own samples, especially if its target is discovery of new, previously unknown attacks.

For that, a number of acquisition systems are required. However, the major components of acquisitions consist of Internet crawlers (input suspicious URLs, output analysis of URL and samples acquired by visiting the suspicious URLs), honeypots (server and client, low and medium interaction), spam traps (mailboxes used for a special purpose, where all incoming email messages are considered either malicious or spam), and sample upload systems that can be used by internal and third-party consumers of the virus lab's services.

In addition to its own collection, a virus lab may decide to use a third party as a source of samples. One such source is *VirusTotal.com*, which holds an exhaustive archive of samples and excellent sample-hunting features.

Analysis

There are currently several unsolved problems in malware research, such as the ability to unpack all custom packers to remove all layers of obfuscation. This is why constant development of analysis systems and analysis techniques is required to keep pace with the techniques used by malware writers to evade analysis, detection and successful malware classification. Nevertheless, certain types of systems are required from the beginning.

The first group of analysis systems are automated analysis systems (Figure 3). They are constantly being developed, and a dedicated team of developers is required to maintain a high analysis success rate. However, there are some free and commercial third-party solutions for automated analysis systems geared towards implementation in virus labs.

Cuckoo Sandbox, an open-source dynamic automated analysis system, is a popular choice often used as a basis for the development of in-house dynamic automated analysis capabilities. The disadvantage of *Cuckoo*'s popularity is that many malware samples do their best to detect its sandboxing environment and avoid displaying any malicious functionality.

The Workflow component is perhaps the most important component of the lab. It is tasked with driving the workflow

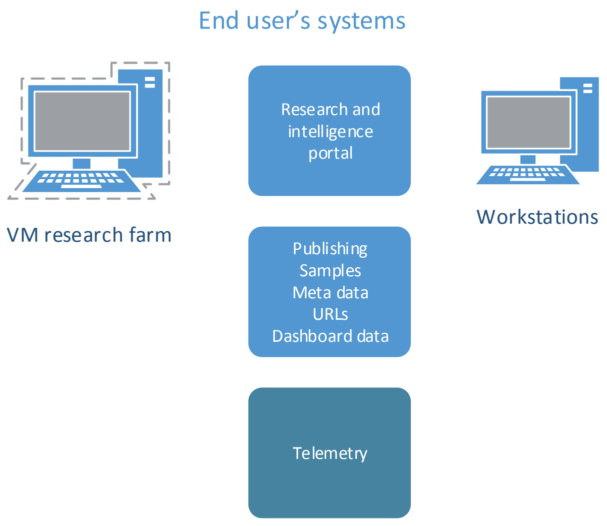

Figure 4: Various end-user systems.

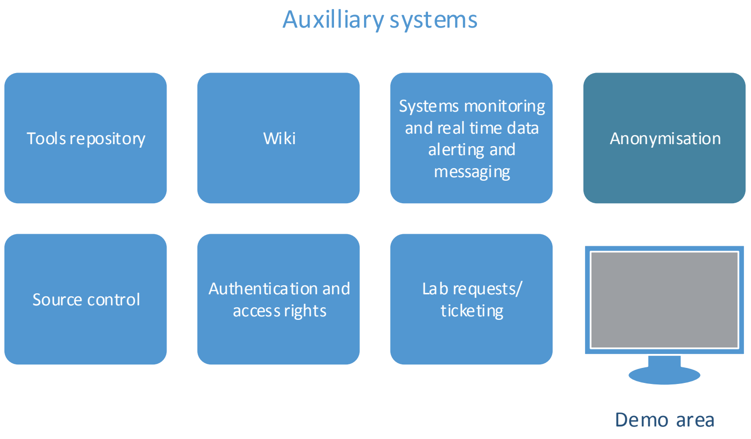

Figure 5: Various important auxiliary viruslab systems.

of sample analysis and coordination of analysis systems as well as collection of results. Typically, it is built on top of a message queuing system integrated with a decision engine to synchronize and dispatch tasks that emulate the manual analysis process. The workflow system acts as a glue between all independent analysis systems. This allows for scaling and synchronization to process almost any required number of daily samples while allowing researchers to use individual systems so that their analysis jobs are run before any other automatically scheduled tasks.

Two last pieces of the analysis systems puzzle, which became important to malware research relatively recently, are automated classification/clustering and data analytic systems. Processing large numbers of samples produces huge amounts of metadata, which remains just data unless classification and analytics systems (together with malware researchers) produce meaningful information to understand and address threats. Analytics and mining systems are required to find similar samples, classify them into clusters and extract previously unknown facts used for protecting the organization from similar attacks. Good clustering allows the virus lab to quickly classify incoming samples simply on the basis of the results of automated analyses.

End-user systems

Although it is possible for researchers to access individual systems to retrieve data, create new processing jobs and analyse samples, the real power lies in consolidating and correlating available data in a single UI accessible to team members and external users on a need-to-know basis. This is what we refer to when we talk about end-user systems (Figure 4).

The Research and Intelligence portal is the main component of end-user systems. Its role is to consolidate information about the overall performance of the virus lab in a dashboard, allowing users to drill down and investigate information about individual samples or a set of samples linked by a single security incident.

The Research and Intelligence portal also allows researchers to reach a farm of VMs suited for individual research tasks, based on the sample file type and the appropriate analysis environment.

Another group of end-users are those external to the team and the company. In their case, the virus lab's output consists of malware collections/samples for sharing as well as near-real-time information about new outbreaks and incidents

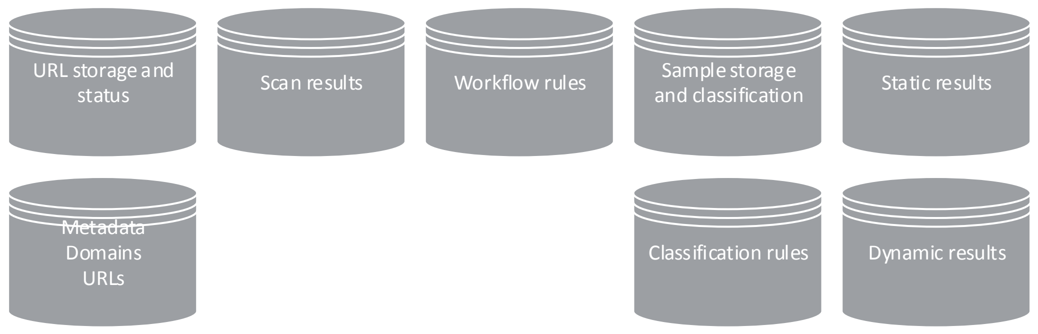

Figure 6: Major storage and database systems.

and the most commonly encountered malware families. The export format should use one of the industry standards such as STIX, MAEC or OpenIOC.

Auxiliary systems

The last category consists of systems important to the functioning of the lab that simply do not fit directly into any of the previous categories (Figure 5).

Systems in this group provide various functionalities, such as repositories for tools or source code control systems used by team members for versioning of YARA [7] rules and of various internally developed analysis utilities.

Auxiliary systems also include systems for controlling authentication and access rights, systems for monitoring and managing other labs' systems, a ticketing system for team requests, and an external-facing anonymizing system that hides the lab's identity from research targets.

Storage and databases

Underlying all these systems is a requirement for massive amounts of hard drive and database storage for storing samples and metadata; storing telemetry data; importing, exporting and storing various partner samples collections; and storage of analysis results.

Figure 6 shows the main storage and database systems that support the lab's processing systems.

It is important for all storage systems to have the ability to be distributed and scalable with high availability, as they have to be used by all other lab systems and end-users.

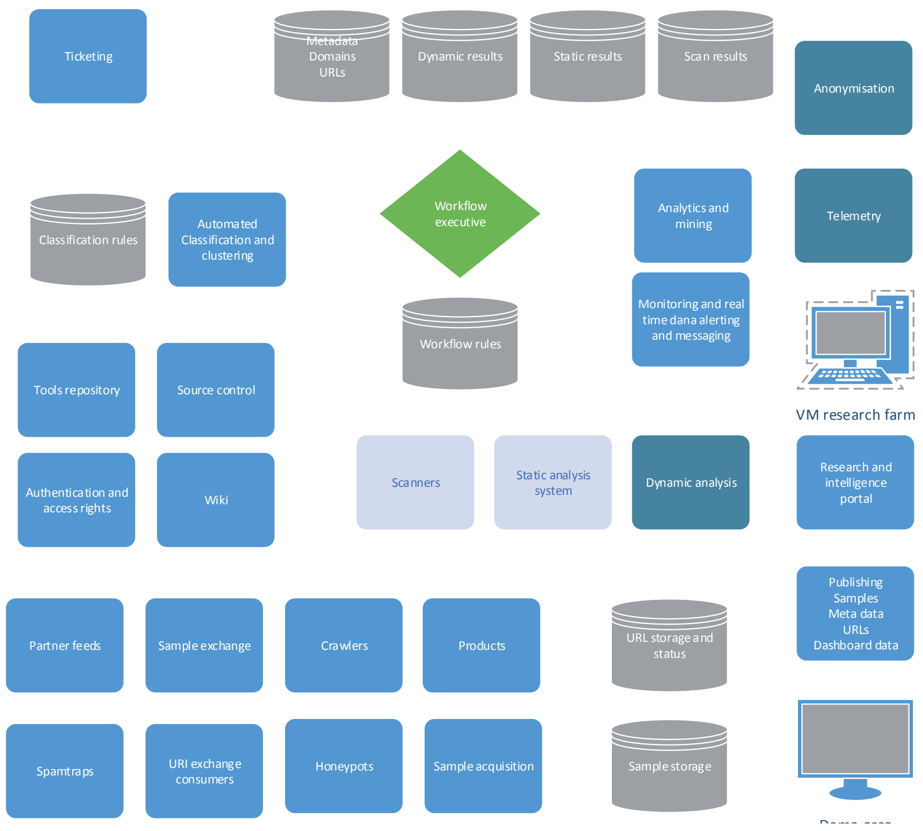

Figure 7: Complexity of the overall set of systems in a virus lab.

The systems in combination

Overall, a fully functional virus lab is a complex system, ideally consisting of loosely coupled systems working together to collect, analyse and research new threats; process existing threats; and convert available data into security intelligence used by internal and external users and systems.

Interlude: If the virus lab was a human

If we think of a virus lab as a living being, we can understand how it has its own needs. Just as in Maslow's hierarchy of human needs [7], basic needs must be satisfied before higher-level needs can be considered. This pyramid may serve as a guideline to the various stages of lab development and their priorities.

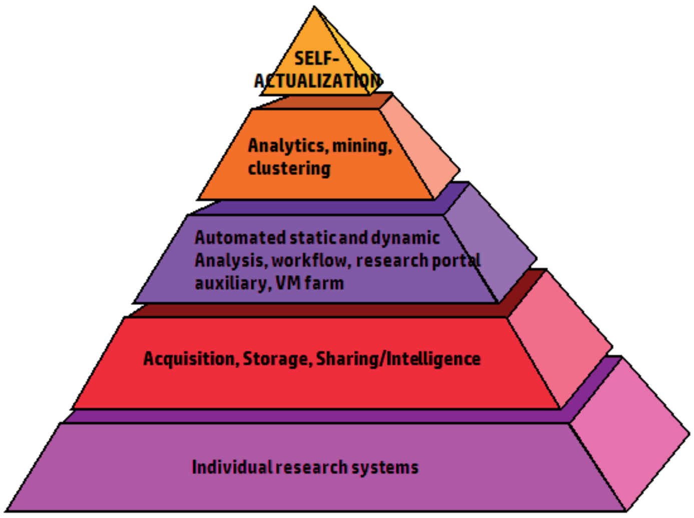
SELF-ACTUALIZATION
Analytics, mining, clustering
Automated static and dynamic Analysis, workflow, research portal auxiliary, VM farm
Acquisition, Storage, Sharing/Intelligence
Individual research systems

Figure 8: A malware research lab's Maslow hierarchy of needs.

A lab starts with individual research systems and grows to self-actualization by choosing building blocks correctly to reach the higher stages in the hierarchy of needs.

LAB DELIVERABLES

Activities of any production system, including a virus lab, would be pointless without a set of deliverables. While in an anti-virus vendor's lab the main deliverables are detection signatures and other data used by the company's product, an enterprise virus lab has the luxury of choosing its deliverable formats, as long as they are useful and increase the security posture of the organization and the malware research community.

YARA rules

YARA is a tool aimed at helping malware researchers identify and classify malware samples [8]. With YARA, descriptions of malware families can be created based on textual or binary patterns. Although a very simple tool with relatively limited functionality, YARA has become a *de facto* standard for rapid response malware detection in organizations and for sharing detection information between independent malware research teams.

The community around YARA is lively, with several YARA rule repositories [9] and rule exchanges [10]. These repositories can help research teams kick-start their usage of YARA, both for the creation of rules and for the detection of malicious activities inside their organizations.

YARA's extensibility is very good, and a team can create auxiliary modules to help with threat detection in their environment. Perhaps the best known set of extension modules is exposed through the malware hunting interface of the *VirusTotal Intelligence* service [11]. This module exposes file type, tags, detection names and other common searching conditions to the YARA rules writer.

With the lack of a more powerful content description language and content-matching utility among the free or open-source options, YARA became by default an indispensable tool in the malware researcher's toolbox.

IDS signatures

Snort [12], Suricata [13] and Bro IDS [14] signatures are other common ways of sharing actionable intelligence about malware. When deployed in a production environment, these systems can point to internal network segments and individual endpoints affected by a malware attack. IDS signatures match on packet payload contents as well as TCP/IP packet header fields. They may be able to prevent download activities as well as some C&C connection attempts, especially if the packet content is unique to the malware family (e.g. a specific bot HTTP User Agent header).

Both YARA rules and IDS signatures have an issue of trust, especially if they are automatically generated by a threat-intelligence sharing framework such as MISP [15]. Before such rules are applied, they should be tested in a non-production environment, which is not always available to a research and response team. That is why deployment of production-quality IDS rules written and tested by a team of experienced IDS researchers, such as the Emerging Threat ETOpen ruleset [16], is preferable to deploying automatically generated rules.

Threat intelligence

The emergence of threat intelligence vendors and the development of the threat intelligence industry were largely driven by the lack of detailed information about existing and newly discovered threats. The volume of newly discovered threats and their complexity made so-called virus encyclopedias containing detailed descriptions of individual malware samples infeasible, or rather impossible, due to the lack of resources needed to analyse every single malware sample.

These encyclopedias are still maintained by traditional anti-virus companies, but rarely contain more than just a description of malware behaviour observed by the company's automated analysis systems. They now serve as landing pages to which users are redirected when their product alerts them to the presence of a malware family.

Today, threat intelligence is expected to be a major deliverable of any virus lab, either in the format of manually written or automatically generated descriptions of observed threats.

Machine-exchangeable information

That said, the type of information usually present in malware encyclopedias still has value, especially for larger enterprises that have decided to build a strong data analytics and correlation capability (e.g. by using an SIEM framework).

With the abundance of threat intelligence data, it became clear that automated, machine-readable exchange formats are required. These formats can be generated and shared between existing threat intelligence frameworks and trusted research partners, in a fashion similar to sample and URL exchanges that already exist within the anti-virus industry.

Semi-structured formats

JavaScript Object Notation (JSON) is a simple, widely adopted, semi-structured and humanly readable data format. The format is native to several NoSQL database management systems as well as to threat intelligence frameworks and existing automated malware analysis systems. For example, the *Cuckoo Sandbox* can easily be configured to export the analysis data in JSON format for further storage, analytics, and sharing.

Structured data exchange formats

Recently, a lot of work has been invested in defining several standards, often XML-based, for describing and exchanging information about IOCs, malware behaviour and attack patterns.

The most popular structured-data formats supported by threat intelligence frameworks are:

- Malware Metadata Exchange Format (MMDEF and MMDEF-B) [17], created by the IEEE ICSG Malware Working Group for the purpose of augmenting shared malware samples with additional structural and behavioural metadata.
- Trusted Automated eXchange of Indicator Information (TAXII) [18], a MITRE standard for machine exchange of threat indicators. The recommended format for the indicators is STIX.
- Structured Threat Information eXpression (STIX) [19] is a collaborative community-driven effort to define and develop a standardized language to represent structured cyberthreat information. It strives to be flexible, extensible, automatable and as humanly readable as possible.
- Cyber Observable eXpression (CybOX) [20], a schema for the specification, capture, characterization and communication of events or stateful properties observable in the operational domain. A wide variety of high-level cybersecurity use cases rely upon such information, including event management/logging, malware characterization, intrusion detection, incident response/management, attack pattern characterization, and so on.
- Malware Attribute Enumeration and Characterization (MAEC) [21], a standardized language for encoding and communicating high-fidelity information about malware based upon attributes such as behaviours, artefacts and attack patterns. It aims to reduce potential duplication of malware analysis efforts by researchers and to allow for the faster development of countermeasures by enabling the ability to leverage responses to previously observed malware instances.
- Common Attack Pattern Enumeration and Classification (CAPEC) [22], a dictionary and classification taxonomy of known attacks that can be used by analysts, developers, testers and educators to advance community understanding and enhance defences.
- Open Indicators Of Compromise (OpenIOC) [23], designed to enable *Mandiant*'s products to codify intelligence to rapidly search for potential security breaches. *Mandiant* has standardized and open-sourced the OpenIOC schema and released tools [24] and utilities to allow machine-to-machine sharing of threat information.

Often, there is very little to choose between various data exchange formats, but it is worth keeping in mind that the virus lab will likely have to be able to consume data conforming to more than one of the above formats and to be able to export data in at least one of them.

BEST PRACTICES FOR DEVELOPING LABS SYSTEMS

Agile systems development principles

Agile principles as defined by the Manifesto for Agile Software Development [25] can, with some modifications, be applied to the development of systems.

At the INCOSE UK ASEC conference in 2012 [26], an *IBM* researcher modified the Agile manifesto and proposed how it can be applied to systems engineering. Agile systems engineering principles are designed to respond well to changing requirements. They focus on delivering demonstrable capability as the primary measure of progress. These principles can easily be applied to the development of virus lab systems.

Here are the top 10 high-level Agile Systems Engineering principles as proposed by Hazel Woodcock [26], with a few additional virus lab-related comments:

1. The highest priority is to satisfy the customer through early and continuous delivery of demonstrable system capability. Having some capability is clearly better than waiting for a fully functional deployment of all virus lab functionality.
2. Managed change to requirements is welcome, even late in development. Agile processes harness change for the customer's competitive advantage. This is also particularly true as virus lab functionality and system features need to adapt constantly to keep up with the latest threats.
3. Deliver actual or modelled functionality frequently, from a couple of weeks onwards, with a preference for the shorter timescale. In a virus lab, the best practice is to allow malware researchers to model some functionality as a proof of concept. Eventually, the development team should implement the feature in a production environment using tried and tested development and quality assurance methods.
4. Business people and the project team must work together daily throughout the project. This process happens regularly in the lab, with the business people in that situation being the malware researchers and experts working together with the systems development team – in some cases being an intrinsic part of the development team.

5. Build projects around motivated individuals. Give them the environment, support what they need, and trust them to get the job done. The field of malware research does not lack motivated individuals. If nothing else, their eagerness has sometimes to be managed to avoid feature drag in the development process.
6. Use the most efficient and effective methods of conveying information to and within a development team, from face-to-face discussion to social business tools and collaborative design tools. It is well known that the need for constant communication flow is proportional to the geographical distance of the team members. Many malware researchers work in globally distributed teams where these requirements are well known, but this principle should not be forgotten.
7. Demonstrable capability is the primary measure of progress. This is the key point for building any system. Demonstrable capability only allows users (researchers) to work more efficiently, but also gives better feedback on the existing demonstrable working features as well as proposing new features.
8. Agile processes promote sustainable development. The sponsors, project team and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential. Malware researchers are not always masters of simplicity. Of course, the key is to try to apply just the right level of complexity to solve a problem with acceptable quality. The KISS principle has been an old friend of both malware researchers and system developers working on the virus lab's systems.

Additional guidelines

In addition to the above Agile principles, we add a few to consider when designing a lab. They take into account the requirement for elasticity of computing power constantly required in the lab – moving from acquisition of samples over static and dynamic analysis and analysis of generated data.

These additional principles were inspired by design principles for web applications running in cloud environments [27]. These cloud applications have been designed from scratch by keeping the advantages and disadvantages of the cloud computing model in mind. Many of those can be applied to the design principles of a virus lab.

1. Start simple with easy wins. The first stage in the malware research cycle is the acquisition of samples, which makes it tempting to start building your own network of honeypots. However, building an intricate, geographically spread modern network of honeypots takes time and yields limited returns. On the other hand, reserving budget for a third-party service such as *VirusTotal* will allow researchers to dig immediately into a wealth of information without even having to build their own storage and automated analysis systems. For individual research, including some trend-spotting, this may be the only thing needed for some time. If a simple internal static analysis framework and malware storage system is required, the *Viper* [28] framework may be the best place to start.
2. Build to scale. Once the decision to build a fully functional lab is made, a cloud platform such as *OpenStack* [29] or *CloudStack* [30] should be considered to allow for horizontal scaling of available computing resources. For example, a Research and Intelligence portal or an automated analysis system needs to be designed to allow for simple addition of new resources without taking the overall system offline. Each of the virus lab's components should be stateless and able to function independently, as is often the case with components of mature virus labs – although that is often simply by chance and not by design.
3. Use auto-scaling of computing resources for adaptable computing power. Auto-scaling allows for programmable maximization of CPU utilization by automatically assigning new computing instances when and where they are required. For example, if the daily load of samples is particularly high, it is possible to auto-scale computing instances so that maximum capacity is dedicated to this task, with only a minimum going to data analytics and clustering and vice versa. This is an important cloud mechanism that can be applied to virus labs where workload can vary from day to day in different functional areas of the lab.
4. Design for failure. Traditional approaches often equate reliability with preventing failure. Cloud-based systems are different, so the design approach has to be different too. In the cloud, one is dealing with greater complexity and more interdependencies – thus more opportunity for failure. Because of that, virus lab services need to be designed to contain failures and recover from them quickly.
5. Harness the power of community. Every lab should be a good member of the malware research community by sharing the knowledge, tools, malware samples and threat intelligence with other community members. By building trust between community members, more and more knowledge will become available to the team. This is why implementation of a malware information sharing framework such as Malware Information Sharing Platform (MISP) and plugging your own systems into it may be a good first step for beginning to contribute to the community. The implementation of MISP may also be useful to established virus labs of traditional anti-virus vendors looking for a better understanding of threats encountered by the community.

CONCLUSION AND FURTHER DIRECTIONS

A growing number of smaller teams require virus labs and systems for malware processing. The requirements of those teams differ from the usual requirements of traditional anti-malware virus labs. This paper provides an introduction into virus lab components and proposes design and development principles for teams with a need for implementation of virus lab functionality.

The implementation of a fully functional lab is a moving target, requiring developers to quickly address changing requirements. Agile development principles are well-suited to the task, even for complex systems engineering such as that required by virus labs.

If developing from scratch, designs of large-scale, cloud-aware applications should be studied and taken into account to provide the virus lab with a flexible environment that can scale and adjust to daily changes in processing loads. Assuming the trend of increasing volumes of newly discovered threats continues, it is likely that virus labs will require the flexible computing and storage environments only available in large cloud deployments. A good design that can fit cloud environments without major changes is required.

The paper provides a set of guidelines that should be followed during the process of building a fully functional malware research lab in a relatively short time, together with a set of near-ready building blocks for each of the lab's functional areas.

Overall, building a virus lab is an extremely complex undertaking, and the team that builds one should be prepared for a constant development process. Continuous development of new modules and features will be required in addition to a significant proportion of maintenance and operations tasks necessary to keep the system performing and adapting to new threat techniques and increasing threat volumes.

REFERENCES

[1] Computer Virus Timeline. http://www.infoplease.com/ipa/A0872842.html.

[2] Kephart, J. O.; Sorkin, G. B.; Arnold, W. C.; Chess, D. M.; Tesauro, G. J., White, S. R. Biologically Inspired Defenses Against Computer Viruses. Proceedings of IJCAI '95, Montreal, August 19–25, pp.985–996, 1995.

[3] Kephart, J. O.; Chess, D. M.; White, S. R. Computers and epidemiology. IEEE Spectrum, Volume 30, Issue 5, May, 1993.

[4] Bussa, T. The Future of Fighting Viruses: A History and Analysis of the Digital Immune. 2002. http://www.giac.org/paper/gsec/706/future-fighting-viruses-history-analysis-digital-immune-system/101594.

[5] Binde, B. E.; McRee, R.; O'Connor, T. J. Assessing Outbound Traffic to Uncover Advanced Persistent Threat. SANS, 22 May 2011. http://www.sans.edu/student-files/projects/JWP-Binde-McRee-OConnor.pdf.

[6] AV-TEST malware statistics. http://www.av-test.org/en/statistics/malware/.

[7] Maslow's hierarchy of needs. http://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs.

[8] YARA in a nutshell. http://plusvic.github.io/yara/.

[9] Yara Rules. http://yararules.com/.

[10] Yara signature exchange google group. Deep End Research website. August 2012. http://www.deependresearch.org/2012/08/yara-signature-exchange-google-group.html.

[11] Virus Total Intelligence. https://www.virustotal.com/intelligence/.

[12] Snort. https://www.snort.org/.

[13] Suricata. Suricata IDS. http://suricata-ids.org/.

[14] The Bro Network Security Monitor. Bro NSM. https://www.bro.org/.

[15] Malware Information Sharing Platform. MISP. https://github.com/MISP.

[16] ETOpen Ruleset. Emerging Threats. http://www.emergingthreats.net/open-source/etopen-ruleset.

[17] About the ICSG malware metadata exchange format working group. June 2015. https://standards.ieee.org/develop/indconn/icsg/mmdef.html.

[18] TAXII. https://taxii.mitre.org/.

[19] STIX. https://stix.mitre.org/.

[20] CybOX. https://cybox.mitre.org/.

[21] MAEC. http://maec.mitre.org/.

[22] CAPEC. http://capec.mitre.org/.

[23] OpenIOC. http://www.openioc.org/.

[24] IOC Editor. Mandiant. http://www.mandiant.com/resources/download/ioc-editor/.

[25] Manifesto for Agile Software Development. http://agilemanifesto.org/.

[26] Woodcock, H. The Agile Manifesto reworked for Systems Engineering. https://www.ibm.com/developerworks/community/blogs/07f4f478-c082-4196-8489-e83384d85a70/entry/agile_se_manifesto?lang=en.

[27] Open Data Center Alliance Best Practice Architecting Cloud-Aware Applications Rev. 1.0. http://www.opendatacenteralliance.org/docs/architecting_cloud_aware_applications.pdf.

[28] Guarnieri, C. Time to do malware research right. http://viper.li/.

[29] Open source software for creating private and public clouds. Openstack. https://www.openstack.org/.

[30] Apache CloudStack Open Source Cloud Computing. Apache. https://cloudstack.apache.org/.

[31] Test of references. http://www.company.com.

[32] Introducing JSON. http://json.org\\\/.

APPENDIX A: CATEGORIZED LIST OF COMPONENTS

These suggestions are offered without warranty, but with the understanding that they are potentially suitable as a starting point in the categories in which they're listed. Most of them will suit one or more purposes but may require modifications to fully suit a set of user requirements.

Sample acquisition methods and systems

Honeypots, spamtraps and crawlers

Modern Honey Network (MHN): http://threatstream.github.io/mhn/

Maltrieve: https://github.com/krmaxwell/maltrieve

Thug client-side honeypot: https://buffer.github.io/thug/

Dionea honeypot: http://dionaea.carnivore.it/

Ragpicker crawler: https://code.google.com/p/malware-crawler/

hpfeeds and Hpfriends [the HP in these stands for Honeypot Project, not *Hewlett-Packard*!]: https://github.com/rep/hpfeeds

Maltrieve feeds crawler: https://github.com/krmaxwell/maltrieve

Sample repositories

Virus Share: http://www.virusshare.com

VirusTotal (not free but in widespread use): http://www.virustotal.com

Malware Share: http://www.malshare.com

Malware Zoo: http://zoo.mlw.re

Open Malware: http://www.openmalware.org

ViruSign: http://www.virusign.com

Services and feeds

Combine: https://github.com/mlsecproject/combine

Criticalstack: https://intel.criticalstack.com/

Malicious URLs Tracking and Exchange MUTE: http://mutegroup.org/

Project Honeypot (spam traps): https://www.projecthoneypot.org/

Abusix Spamfeedme (spamtraps): https://spamfeedme.abusix.com/

Shadowserver feeds: https://www.shadowserver.org/wiki/pmwiki.php/Services/Downloads

Bambenekconsulting: http://osint.bambenekconsulting.com/feeds/

Malware Patrol: https://www.malwarepatrol.net/lists.shtml

Anonymizing services

Tor: https://www.torproject.org/

OpenVPN: https://openvpn.net/

Privoxy: http://www.privoxy.org/

Storage and organization

Viper malware storage and static analysis framework: http://viper.li

Aleph Malware Analysis pipeline system: https://github.com/trendmicro/aleph

Automated analysis

Multi scanner frameworks

Opswat Metascan (commercial): https://www.opswat.com/products/metascan

IRMA analysis platform for suspicious files: http://irma.quarkslab.com/index.html

Malice, a framework for building a VirusTotal-like system: https://github.com/blacktop/malice

MITRE's Multiscanner: https://github.com/MITRECND/multiscanner

Multiav scanner wrapper: https://github.com/joxeankoret/multiav

Static analysis frameworks

Mastiff framework: https://git.korelogic.com/mastiff.git/

Viper: http://viper.li

Machete framework (in development at the time of writing, no URL yet)

Dynamic analysis systems

Cuckoo Sandbox: http://www.cuckoosandbox.org/

Cuckoo Accuvant fork: https://github.com/brad-accuvant/cuckoo-modified

Cuckoo Mitre fork: https://github.com/MITRECND/cuckoo

Cuckoo KillerInstinc (better office handling) fork: https://github.com/KillerInstinct/cuckoo-modified

Killerinstinct Cuckoo modules repository: https://github.com/KillerInstinct/community

VMCloak hiding Cuckoo analysis VM configuration tool: http://vmcloak.org/

Cuckoo auto installer (aids with Cuckoo Sandbox installation): https://github.com/buguroo/cuckooautoinstall

Online analysis services

Cuckoo Sandbox online: https://malwr.com

VirusTotal: https://www.virustotal.com

Team Cymru's Malware Hash Registry: http://www.team-cymru.org/MHR.html

Team Cymru's TotalHash.com static and dynamic analysis: https://totalhash.cymru.com/

Integration

ZeroMQ: http://zeromq.org/

RabbitMQ: https://www.rabbitmq.com/

Intellect rule engine: https://github.com/nemonik/Intellect

Celery distributed task queue: http://www.celeryproject.org/

Big Data processing and indexing

Elasticsearch stack: https://www.elastic.co/downloads

Netflix Scumblr (mining the web): https://github.com/Netflix/Scumblr

Hadoop: https://hadoop.apache.org/

Apache Spark: https://spark.apache.org/

Databases

MariaDB: https://mariadb.org/

neo4j graph database: http://neo4j.com/

MongoDB: https://www.mongodb.org/

CouchDB: http://couchdb.apache.org/

Cassandra: http://cassandra.apache.org/

HBase: http://hbase.apache.org/

Visualization and graphing

Davix tools: http://www.secviz.org/node/89

OpenDNS OpenGraffiti: http://www.opengraphiti.com/

Stix Viz, visualization of STIX data: https://github.com/STIXProject/stix-viz

Kibana, Elastic search visualization: https://www.elastic.co/products/kibana

Threat intelligence sharing systems

MISP Malware Information Sharing Platform: https://github.com/MISP/MISP

CRITS Collaborative Research Into Threats: https://crits.github.io/

CIF Collective Intelligence Framework: https://code.google.com/p/collective-intelligence-framework/

Soltra Edge intelligence framework: https://www.soltra.com/

Siemens Django Mantis Intelligence framework: https://github.com/siemens/django-mantis

Threat intelligence online

ThreatConnect: http://threatconnect.com/

HP ThreatCentral: http://go.saas.hp.com/software/threat-central

ThreatStream: https://www.threatstream.com/

FB ThreatExchange: https://threatexchange.fb.com/

Sample sharing systems

Virex virus exchange framework: https://code.google.com/p/virex/

Incident response framework

Fast incident response by Societe Generale CERT: https://github.com/certsocietegenerale/FIR

Machine learning

Amazon Machine Learning: https://aws.amazon.com/blogs/aws/amazon-machine-learning-make-data-driven-decisions-at-scale/

Windows Azure Machine learning: http://azure.microsoft.com/en-us/services/machine-learning/

Distributed R: https://github.com/vertica/DistributedR

Python Anaconda: https://store.continuum.io/cshop/anaconda/

Orange Python Data mining: http://orange.biolab.si/

Weka toolkit: http://www.cs.waikato.ac.nz/ml/weka/

RapidMiner: https://github.com/rapidminer/rapidminer

Preconfigured containers

Docker containers developed by YoloThre.at: http://yolothre.at