Project Blackfin:
Automated Breach Detection Using Intelligent Agents

CONTENTS

ABSTRACT

This paper describes a distributed anomaly detection approach designed to accurately track actions performed by adversaries across a network. The illustrated approach involves running multiple machine learning models on both endpoints and the network as well as on a centralized backend, and includes communication and replication of learned states between these models. The research described in this document was conducted by F-Secure Corporation as the first stage of a multi-year research project, designated “Project Blackfin”. Some of the methods presented in this document have already been implemented in our own dynamic breach detection solutions. The near-term goals of this project include (i) invention of new, more generic methods for detecting adversarial actions, (ii) creation of mechanisms capable of tracking attacker actions across multiple endpoints on a network, (iii) improvement and automation of threat intelligence gathering capabilities, (iv) understanding of how to implement and improve automated response actions, and (v) implementation of mechanisms that are able to perform contextual risk analysis on each endpoint.

INTRODUCTION

Modern-day breach detection strategies commonly rely on gathering and aggregating streams of endpoint and network traffic data, processing and analyzing the data as it arrives, and storing the data in a centralized location for subsequent analysis and audit purposes. Incoming data is processed by algorithms that may include hand-written rules and machine learning models. As new tactics, techniques, and procedures (TTPs) and their indicators of compromise (IoCs) are discovered, detection logic is updated, and may be run against historical data in order to confirm that any newly discovered attack vectors weren't missed.

Due to their nature (which will be explained shortly), centralized detection approaches that rely on the aggregation of collected data have limited ability to correlate events across multiple different sources, and to correlate newly arriving events with historical data, on-the-fly (especially if there is a significant time-delta between those key events). They also struggle to cost-efficiently process the large amounts of data required to detect signs of attacker actions. These solutions are largely designed to quickly identify individual instances of attacker behavior, and are often less effective at tracking an attacker's previous actions across multiple endpoints on a network. When signs of a breach are detected, manual verification steps are often required, and it is then up to a team of humans to respond to the situation. Response actions can include isolating compromised systems, performing forensics to determine actions the attacker performed, and cleaning or re-imaging systems in order to remove the attacker's access to the network.

The methods used by attackers to take control of an endpoint, perform recon, laterally move to other endpoints, establish persistence mechanisms, exfiltrate data, and establish uni- or bi-directional command-and-control channels employ a variety of mechanisms, the detection of which requires examination of different types of system information, often from multiple endpoints. Information relevant to discovering an action an attacker performed may include process creations, network connections, logon events, named pipe creations, module loads, file accesses, registry accesses, and system log entries, to name a few. Most of the traces left by adversarial actions blend in with naturally-occurring activity on an operating system or network. As such, not only are many different mechanisms needed to detect all of the possible actions that could be performed by an attacker, but those actions must be accurately differentiated from the thousands of benign events on a system, series of systems, or network.

Events generated by actions that attackers perform are typically very sparse. For instance, the creation of a process on a target endpoint, a network connection between the source and target, and perhaps the creation of a named pipe on the target endpoint may be all that might indicate a lateral movement action has been performed by an attacker. Very similar events will be generated when a sysadmin performs a remote administrative task on a machine in their organization. There are several ways in which the action of an attacker may be differentiated from the action of the sysadmin – one might identify that the source

IP address doesn’t belong to one of the sysadmins’ machines, that the remote execution involved a tool that isn’t part of the organization’s administrative toolkit, or that the command run on the target machine wasn’t a command typically issued by an administrator. In all cases, prior knowledge, and even intimate familiarity of the often poorly documented target environment is required to spot adversarial activity with adequate confidence (a list of legitimate sysadmin IP addresses, a list of known administrator tools, and so on).

When an attacker performs an action on a system, it will likely generate a single, or very small number of events on one or more systems that will arrive alongside thousands of others from adjacent endpoints on the network. As such, a system designed to aggregate and analyze incoming events from multiple endpoints across a network, would require either very specific rules, or complex cross-correlation of data from multiple endpoints in order to detect that action, as it happens. Hence, systems built on the premise of analysing streams of incoming events often fail to alert properly - they're either too general and miss things or too careful and flood the administrator with false positives. In order to obtain actionable information from such a system, the creation and maintenance of a set of rules designed to suppress false positives is required. This is both cumbersome and time-consuming.

Creating automation designed to detect the subtle actions of an attacker, across multiple machines on a network, over a period of many days, weeks, or even months, is a complex endeavor. Consider the fact that, once an intrusion is detected, it usually takes a team of experienced forensics experts a considerable amount of time to assemble a timeline of what the intruder did, and what other systems they visited during the breach. In order for an automated system to enumerate the adversary’s prior actions, it would need to run complex logic over an arbitrary amount of historical data collected from potentially thousands of separate endpoints.

Anomaly detection methods seek to establish a baseline for behavior in an environment and alert when an event occurs that doesn't fall within that baseline. For instance, anomaly detection applied to traffic on a company's network could be used to build a baseline understanding of which IP addresses communicate directly with each other, and alert if two IP addresses that don't normally communicate directly suddenly start doing so.

In this paper, we detail a distributed anomaly detection approach for accurately tracking actions performed by adversaries across a network. The described approach involves running multiple models on both endpoints and the network as well as on a centralized backend, and includes communication and replication of learned states between these models. This methodology enables:

- More expedient and efficient detection of threats, leading to faster, more accurate response actions.
- Threat detection methods at the endpoint-level and mechanisms to share understanding of relevant anomalies across the systems on a network.
- Active utilization of the state of an organization, locality, or subnet (as opposed to a single machine) to improve detection capabilities.

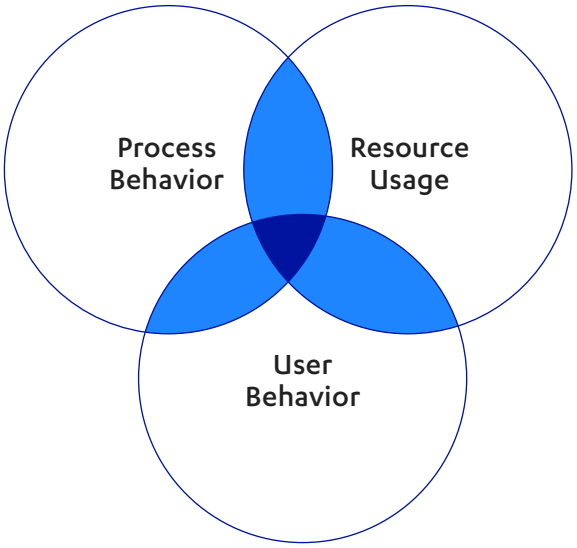
- Automated methods for discovery of lateral movement actions.
- Automated methods enabling understanding about how and where an attacker is operating on a network.

The intention of this document is to describe research conducted as part of this project, suggest additional mechanisms that may be used to augment our described methodologies, and enumerate the potential benefits and pitfalls that accompany our described approaches. This multi-year research project, under the moniker “Project Blackfin” is being conducted by F-Secure, and involves collaboration between our in-house engineers, researchers and data scientists, and academic partners. Some of the described mechanisms are already in use in our products.

ANOMALY DETECTION METHODOLOGIES - A CYBER SECURITY PERSPECTIVE

One common approach for detecting anomalous behavior involves the creation of a baseline of typical behavior. This is done by profiling a targeted object, system, or user, for a period of time. In practice, it is possible to profile a wide variety of data sources in an organization. These data sources range from low-level operating system events, to behavioral patterns exhibited by users, to interactions between objects on a network. Depending on the use case, profiling may be performed on an endpoint, on a server, on the network, or across aggregated streams of data from multiple endpoints.

In order to detect an attacker, it is necessary to understand the data sources that might provide visibility into their actions. One way to approach this is to separate the problem space into three categories.



Process behavior – is a certain program behaving normally?

Patterns found in e.g. executable process creations, parent-child relationships between processes (which executable process on a computer launched which other process), and actions performed by the processes themselves – such as module loads and system resource accesses are useful for discovering many of the tactics used by attackers, such as exploitation of unknown vulnerabilities, or trojanization of an executable.

Resource usage – is an object being used normally?

Anomalous resource usage patterns (CPU load, memory usage, disk reads/writes, network reads/writes, etc.) can uncover malicious actions such as ransomware, cryptocurrency miners, exfiltration techniques, or other custom tasks initiated by intruders.

User behavior – is a user behaving normally?

By profiling logon and logoff events (including from where, and to where), application launches, execution durations, network locations accessed, files accessed (including those on file servers) in the context of a specific user, it is possible to discover anomalous behavior patterns that may indicate the presence of an intruder, data exfiltration, or indeed a user behaving maliciously.

It should be noted that the above three-fold representation is just one of the many possible ways of working with anomalous behavior.

A note on data representation and machine learning approaches

Each available data source can be potentially organized and represented in a variety of different ways. For instance, data related to executable processes can be represented as sequences, node-edge graphs, timelines, various aggregates, or even simple counters. There are also many methods available for analyzing or modelling each data source, depending on the chosen data representation, and those methods can also be ensembled and combined in many ways. When considering models designed to run or even be trained on endpoint machines, resource usage is of concern. More resource-intensive methodologies may be more suited for use on backend systems than on endpoints.

When considering current state of the art in artificial intelligence and machine learning, deep learning approaches are often the most cited due to their deserved popularity. Deep neural networks have achieved unparalleled results in many challenging tasks, and will undoubtedly continue to do so for years to come. While we certainly have nothing against deep learning (and are, indeed, using it ourselves), we note that it is far from the only viable approach for implementation of effective machine learning-based solutions. State of the art review indicates that other approaches are often more effective for dynamic attack detection. However, some recent NLP and object recognition methods could prove useful in this domain. We have experimented with many approaches and chosen methodology that best fits the problems we are trying to solve.

In some domains, maximization of classification accuracy is the main objective. For these domains, approaches that generate, on average, the most accurate point estimates of the posterior distribution are desired. Applications such as image recognition often fall into this category – traditional tests run on academic datasets have focused on reaching and surpassing human accuracy.[1] Complex classifiers, such as deep neural networks, are able to create very flexible decision boundaries when trained with a large number of samples, and hence are very good solutions for these kinds of problems.

---

1 As a side note, how one can measure accuracy beyond human-level in, for example, image classification, turns out to be an interesting philosophical conundrum. If a human cannot determine what a handwritten digit should be, how is it possible to evaluate performance and results of a machine learning classifier on that sample?

In other domains, the outcomes of decisions made by models are far from equal. In some cases, mistakes can be very costly – or even fatal. In fields such as healthcare or cyber security, it is not enough to simply choose which alternative is the best fit – to ensure we make the right decisions we should aim to understand how certain we are about the decision we are making. Modelling approaches that are able to consider the uncertainty of decisions, provide an estimate of the posterior distribution, and that allow encoding of prior knowledge into the model provide a clearer understanding of the risks related to the final decision. Modelling approaches like these may not always be the best "on average", but given the domain, are best suited for both making critical decisions and understanding their outcomes.

Models designed to be used for dynamic attack detection need to perform well in environments that are subject to constant change (for instance, where software and even hardware may change on a fairly regular basis). Models that are able to perform tasks in a changeable environment are also likely to perform accurately for longer periods of time. A model's robustness against attacks is also important in the presence of adversarial parties. As such, simpler, more robust models are better suited to combat common adversarial attacks, such as data and model poisoning, especially when federated or distributed learning approaches are also being used. Simpler models can also bring other benefits, such as explainability of decisions, which can greatly ease identification of false positives and false negatives. The ability to understand the impacts of individual data points in complex models is a daunting task but some problems are only solvable with more complex, and thus less interpretable models – it is often a trade-off, but one that should be made consciously. In real life often, no silver bullet exists.

Combining anomaly detection models to detect specific phenomena

By combining the outputs of multiple different models associated with each of the categories alluded to earlier, a contextual understanding of what is happening on a system can be derived, enabling downstream logic to more accurately predict whether a specific event or item is anomalous, and if it is, if it is worth alerting on. This approach enables generic methodologies for detecting attacker actions (or sequences of actions), without baking specific logic into the detection system itself. A system built within these design parameters should be able to accurately answer specific classes of questions such as:

- Is a certain program behaving normally?
- Is a certain object being used normally?
- Is the user behaving normally?

Areas of intersection between two or more of the above categories represent methods for detecting anomalies that can be addressed from more than one point of view.

We will now illustrate these concepts with a number of real-world examples.

Example: discovery of rare, potentially harmful actions on a network

As security researchers, intuition tells us that an instance of winword.exe launching cmd.exe is likely a signal of malicious activity on a system. By utilizing a simple model that gathers statistical information about process creation events on a system, it should possible to identify parent-child relationships between executables indicative of malicious activity, in a generic manner. Here, we illustrate a method for scoring process creation events in order to ascertain whether they are suspicious. A process creation event is considered anomalous if:

> The child process is common - the child is often spawned (by a parent process of any name) **AND** the parent process is common - the parent process often opens new processes (of any name) **AND** the parent-child combination is rare - the process of this particular name rarely, if at all, opens a child of this particular name.

Applying this logic, with some simple normalizations, to our initial example, winword.exe and cmd.exe are both common, but the winword.exe - cmd.exe pair is rare, and thus a strong signal of anomalous activity. As it turns out, this process pair is often seen in spear-phishing attacks. Payloads generated by Metasploit work in a similar fashion. This methodology represents a generic mechanism for quickly identifying new, unknown malicious samples - especially ones that have been purposefully crafted for targeted attacks. For instance, if winword.exe launches cmd.exe on a computer on the network, the document that caused this action would be of interest. This technique has already proven itself to us in the field.

Our described methodology for detecting anomalous process creation events is a good example of a system that can easily utilize federated learning mechanisms to distribute learnings across multiple endpoints on a network. Federated learning is also applicable to several of our other described methods.

Federated Learning

Federated learning is a machine learning setting where the goal is to train a high-quality centralized model with training data distributed over a large number of clients. Smartphone predictive text input is a good example of a federated learning system. Predictive text input suggests word completions as you type into a smart phone keyboard. Suggestions are provided by a machine learning model running on the phone itself. When the user of the phone selects a suggested word, or ignores the suggestions and completes a word manually, the model on the phone learns that user's preferences. Over time, the local model gets better at suggesting words the user wants. Each phone on the network periodically submits its local model to a central server, where the parameters of the submitted model are used to update a central model. This central model, in turn, is periodically deployed back to all phones on the network, where it is used to update or augment local models. Over time, the central model learns from millions of users and, due to the update mechanism, improves text prediction models running on each local device. A new phone will receive a trained model that serves as a useful generic baseline until that local model learns from the user's behavior.

Example: Detecting ransomware-like behavior

Ransomware is a class of malware that will, in most cases, traverse a filesystem, encrypting files that match specific criteria (e.g. file types: documents, images, videos, etc.). Ransomware processes exhibit anomalous behavior by virtue of being unique, and usually untrusted processes that sequentially enumerates and modifies files in a filesystem. From the point of view of filesystem-based anomaly detection logic, the behavioral patterns exhibited by a ransomware process are highly irregular (compared to most other processes) and would likely immediately warrant an alert, especially when combined with relevant metadata about the process itself, such as information about the executable's signature, structure, when it arrived on the system, where it was downloaded from, and so on. It is true that some software auto-updaters modify files on disk (as opposed to writing new files). However, such updaters likely run on a fairly regular basis, perform modifications on the same set of files each time, and may indeed be trusted. As such, their actions will likely be recorded as part of a model's baseline, and will thus not generate alerts.

Example: Detecting signs of lateral movement

Detection of lateral movement actions by attackers has traditionally been a difficult task to solve using common breach detection methods. Consider the problem of determining whether a user logon belongs to the owner of a system or an adversary. Every user is unique - they arrive at work at different times, take breaks at different times, go on vacation during different periods, perform different tasks at work, browse different websites, and so on. The only true way to create a meaningful baseline for each user (such that we can accurately detect that it was an intruder who logged into their machine, and not them) is to profile each user individually. Attempting to profile user behavior at an organization- or role-level typically leads to over-generalization, and models that aren't able to understand the nuances of each user's behavior. Generic models like these are useful for learning common shared behaviors between users, such as typical office hours, weekends, and national holidays, and can also learn information about standard software usage and network topology.

In order to have a better chance of finding an attacker, it is beneficial to profile each user separately. However, a baseline of a specific user's activity might not still be enough to determine whether a logon event belongs to an intruder. If we consider an approach that combines the outputs of multiple specialized models, however, making such a determination becomes significantly easier. Consider what happens when the following are used in combination:

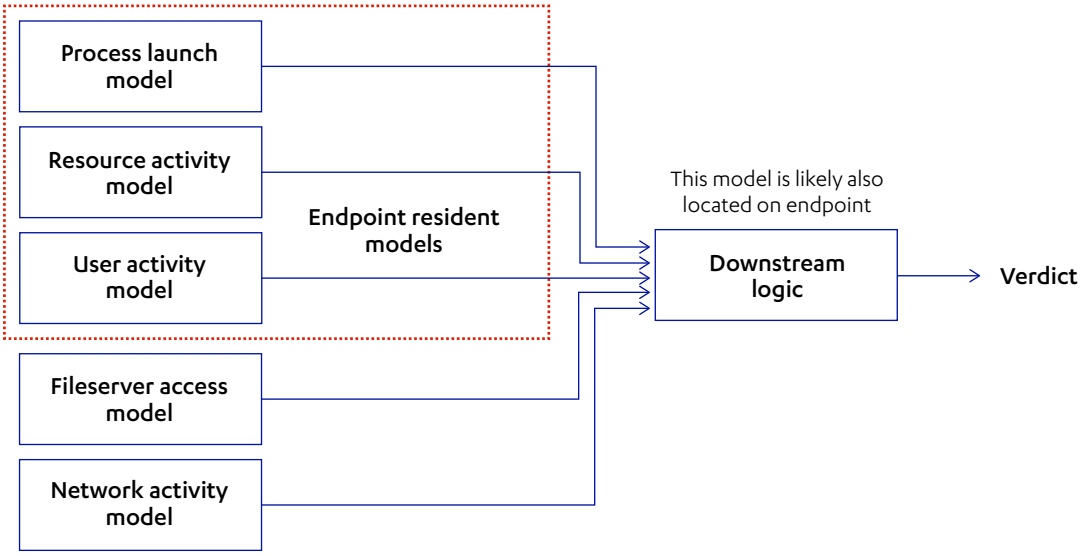
- A model that tracks baseline process launch behavior patterns for a user – when the user account launches and terminates processes, which processes they are, and which processes launched by the user subsequently launch other processes.
- A model that tracks baseline resource activity patterns on the user's system – i.e. CPU load, memory usage, disk activity, network activity, etc.
- A model that captures a user's logon and logoff patterns.

- A model that tracks a user's file access patterns, including accesses to files on servers.
- A model that tracks network traffic patterns across an organization's network (a network sensor or back end model).

Combining the output of the above models allows downstream logic to more easily differentiate patterns produced by an adversary from patterns typically produced by the user of the affected system. If an attacker intrudes on that user's system, they can be detected via a combination of facts, such as:

- The attacker launched a process that the user has rarely or never launched.
- The attacker performed activities on the endpoint during times that the user is typically idle.
- The attacker accessed files (for instance on a file server) that the user has never or rarely accessed before.
- The attacker made network connections to other endpoints that the user has never or rarely made before.

Here is one possible illustrated example of the described detection process:



Depending on the baseline learned by aggregator models, the described approach may be useful for detecting certain other classes of maliciously-motivated user behaviors, such as insider threats. For example, the described methodology could be used to generate alerts if a disgruntled employee decides to start copying a large number of confidential files from an internal server to their local machine, to a cloud service, or to a USB drive over the course of a number of hours or days. User behavior modeling is a deemed a "potentially privacy endangering" activity under EU's General Data Protection Regulation, and hence any functionality designed to detect anomalies in users' behavior would need to accommodate the associated privacy requirements.

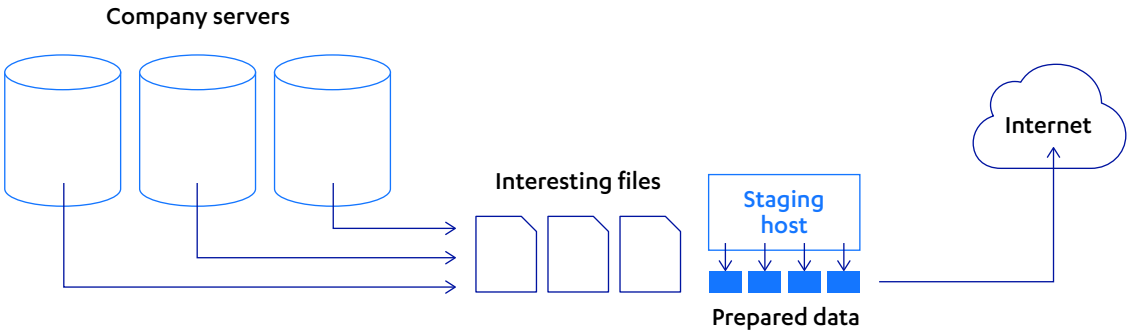
Example: detecting adversarial access to confidential information

Attackers commonly search for and exfiltrate confidential data from an organization once they have access to one or more key systems in that organization's network. In order to do this, an attacker may log into one or more endpoints on the network using accounts that have permission to access required files or directories on a file server. They will then perform operations on the file server such as browsing, searching or copying. The data that is ultimately selected for exfiltration will likely be relevant for espionage, blackmail, or financial gain – e.g. company secrets, source code, customer information, financial records, salary information, strategy plans, etc.

An anomaly detection model running on a file server can be used to learn file access baseline behaviors – which users access which files, when they typically access them, and what operations they perform. Such a model, especially when combined with other types of models described in this document, can be used to form a system capable of detecting signs of an attacker, such as when a user accesses a number of files they haven't accessed before, when a user accesses files from a different endpoint than usual, or when recursive file access operations are performed on a file server. The information gathered from such a system is useful for determining the actions the intruder performed – which files they accessed, if they modified any files, where they copied files to, which endpoints they accessed the files from, which user accounts may be compromised, and so on.

Example: detecting web-based exfiltration mechanisms

A variety of techniques for exfiltration (copying data out of a breached network) exist, most of which are designed to blend with common traffic on the victim organization's network. These techniques can include the utilization of common web services (Dropbox, YouTube, Pastebin, Github), email (often to accounts created on free webmail services such as Gmail), custom exfiltration tools, and via protocol abuse (such as dnscat). A seasoned attacker will likely select a stealthy exfiltration technique based on knowledge gathered from the victim's network. Exfiltrated data is often staged somewhere on the victim's network and then packaged (compressed, encrypted, split into small chunks) prior to exfiltration – in order to avoid detection by data loss prevention solutions.



An anomaly detection model designed to learn process behaviors on individual endpoints can be used to spot some types of exfiltration techniques. For instance, if a web browser process on an endpoint suddenly accesses many files, it may indicate that data is being uploaded to a web service. The output of such a system, when correlated against the output of a network anomaly detection model, can be used to determine which files were exfiltrated, and where the files were exfiltrated to. Certain data exfiltration actions can also be detected by a fileserver-based model that is designed to alert on large unexpected data movements. Finally, the steps used to prepare data for exfiltration appear anomalous on most users' systems, since they typically involve the use of specialized tools and command-line operations.

Example: detection of traits associated with supply chain attacks

Supply chain attacks occur when an attacker compromises one company in order to gain access to a second company or organization that is being supplied by that company. An adversary may choose to use a supply chain attack when their target is too well defended, and thus too difficult to breach directly. Supply chain attacks employ a variety of different methods. Some recent, high-profile supply chain attacks have involved compromising software vendors. In these cases, adversaries introduced malicious code into the vendors' software, and then relied on the fact that users of the trojanized software would update to new versions. Updating software in a timely manner is considered a good security practice.

These recent high-profile supply chain attacks (ShadowHammer, ShadowPad, and CCleaner) demonstrate how adversaries are able to easily compromise systems by abusing the trust implicitly contained in authenticode-signed Windows binaries. Traditional security controls struggle with these kinds of scenarios, since they often whitelist signed binaries in the name of performance and false positive-prevention.

Supply chain attacks that rely on trojanizing software typically seek to stealthily embed malicious code into a binary prior to the signing step. To do this, they may employ one (or more) of the following mechanisms:

- The adversary commits malicious changes to portions of the binary's source code.
- The adversary "patches" malicious functionality into an already compiled binary prior to the signing step (i.e. by modifying bytes in the binary on disk).
- The adversary alters the victim organization's build environment such that malicious functionality is introduced during the binary's build process.

It is often the case that the malicious payload in a binary is executed conditionally – should a certain set of circumstances apply. This is done for a number of potential reasons:

- So that the payload only runs for specific victims or on specific systems (e.g. a malware only runs on operating systems embedded into ATM machines, malware only runs on machines in Ukraine).

- To evade automated analysis processes or to stifle reverse-engineering efforts (e.g. the executable verifies it is not running in a virtual machine, or under instrumentation, or the payload waits for some time before executing in hopes that automated analysis systems have already exited their analysis by the time the payload would trigger).
- To verify that pre-requisite conditions exist on a system – i.e. that the system includes necessary libraries and functionality needed. (e.g. the malware only works on Windows 7 and newer systems).

Malicious payloads often utilize functions provided by system modules. Since it is not guaranteed that these modules were loaded during initialization of the hijacked binary, the code contained in a payload may perform delayed load operations in order to make these functions available. If changes in module load timing patterns between different versions of an executable are detected, this information, in conjunction with other metadata, can be used to determine whether an application has been trojanized.

More generally, the actions of individual executables can be profiled over time in order to generate a baseline of typical behavior. These actions can include file accesses, registry accesses, module loads, process and thread creations, network accesses and so on. If the observed behavior of a new binary significantly differs from profiled behavior, it may be indicative of trojanization.

Sometimes, static analysis of executables, carried out in the backend or locally by a modern anti-malware program, can help recognize threats. Static features and meta-data, such as location in the filesystem, size of binary, headers, strings, relevant hashes, imports, and so on, can be used to compare new versions of binaries against earlier ones and – in conjunction with behavior-based indicators – can identify maliciously altered software with high confidence, minimizing the risk of false alarms.

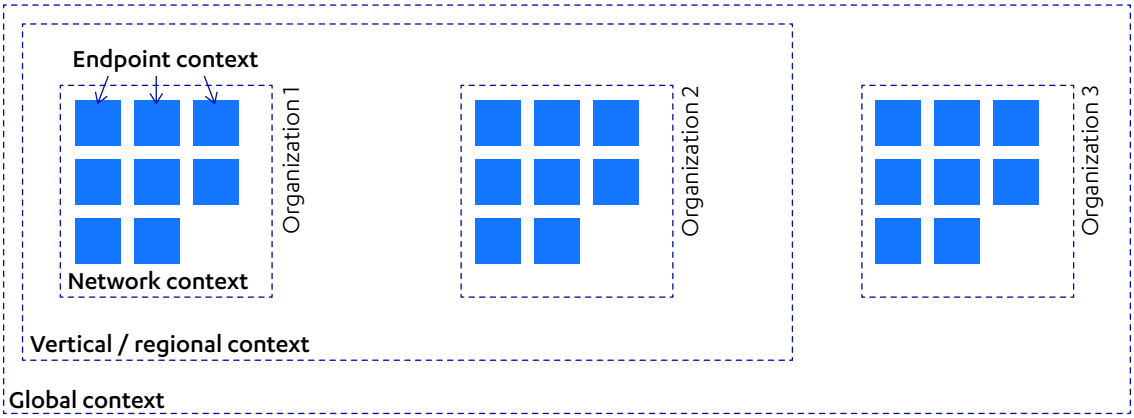
DESIGN CONSIDERATIONS

The examples in this document represent just a handful of detection mechanisms that are being studied as part of this project. As our research continues, we expect to gain deeper insight into the interactions between the described models, and what value such interactions may bring. These insights will help steer future design decisions.

Different models require different training durations in order to learn accurate baselines, and some models are more generic than others. For instance, models designed to detect anomalous executable behavior learn similar representations on all endpoint systems. As such, generic models of these types can be pre-trained and deployed on new endpoints, where they become immediately effective. Models designed to baseline user behavior, however, require time to learn each individual's actions. In this case, generic, pre-trained user behavioral models can be deployed to new endpoints, since they will already have learned common behaviors across users in the organization such as work hours, public holidays, and so on. These pre-trained models provide a starting point after which the model will, over time, learn the specific behaviors of the owner of the system.

Some of the examples in this document demonstrate mechanisms where the output of multiple models are used as inputs to subsequent models or algorithms. For instance, our described lateral movement detection mechanism involves combining outputs from models running locally on an endpoint with models running on the network and back end. We expect similar mechanisms to be useful for the detection of other types of actions. By mixing and matching different combinations of endpoint-level, network-level, and global-level observations, systems capable of detecting different types of attacks in a generic manner can easily be created.

Mechanisms such as federated learning enable representations learned on one system to be shared with other models on the network. A learned representation is, to a varying degree, useful to other users within the same context. Contexts are sets of users or endpoints, grouped by one or more similar attributes. Contexts can include employee roles, teams, organizations, industry verticals, or everyone (the global context), as illustrated in the following diagram.



In reality, it will often be useful to create multiple overlapping contexts in order to better understand each individual. For instance, contexts may be created for categories such as:

- Financial experts (by virtue of specific software used on their systems, or access to specific network resources that other users don't access).
- Software developers (by virtue of development software running on their machines and the fact that new, unique binaries are often detected on their systems).
- Users that work in the evenings (for instance if shift work is utilized at a company).
- Administrators (by virtue of the fact that they access other machines on the network in order to perform administrative tasks).

Each user's context determines how their local models are augmented, and who their own models are shared with. Threats and anomalies are thus identified on the basis of the combination of these models at each endpoint, which is determined by context. After all, the real meaning and relevance of an anomaly arises from the context in which it is observed. In order to be able to detect relevant threats, one has to be able to select relevant contexts. Additionally, when a threat is detected, the context in which that threat was discovered is then used to determine where else it may exist.

BENEFITS OF DISTRIBUTED BREACH DETECTION

Models running on each endpoint are able to make decisions without the need to stream all collected data to a back end. This helps reduce data transmission, processing, and storage costs, which ultimately makes our solutions more cost-efficient for customers.

Locally-run models are also able to catch threats much more quickly, and cooperate with other systems on the network to detect anomalies (for instance by sharing facts or learnings, or requesting information).

Another important consideration is user privacy regulations, which limit what data can leave customer environments, and in particular what degree of access an attack detection solution has to user behavior data. This can include things such as the contents of documents, and URLs that contain login credentials. Local models can, however, work with this data. The availability of data sources that were previously off-limits to backend-centric detection solutions opens up potentially new detection methodologies or capabilities.

NEXT STEPS

We appreciate that, for the foreseeable future, the processes required to identify and remediate targeted attacks will continue to involve a combination of software-based technologies and human expertise. The near-term goals of this research are thus to:

- Develop new, more generic methods for detecting adversarial actions.
- Create mechanisms capable of tracking attacker actions across multiple endpoints on a network.
- Further improve and automate threat intelligence gathering capabilities.
- Understand how to implement and improve automated response actions.
- Implement mechanisms that are able to perform contextual risk analysis on each endpoint (where each endpoint's risk is determined by actions observed on other endpoints in close proximity).

Improved threat intelligence gathering

Generic mechanisms designed to detect anomalous behavior will inevitably discover new malicious samples and behaviors. These discoveries represent important happenings in threat intelligence, the results of which can be sent to researchers, and used to identify similar threats within the same, or other networks.

Automated response actions

Client-side logic can be implemented to perform actions such as capturing forensic data, or isolating endpoints from the network in response to detected actions from attackers. Endpoint-to-endpoint communications further enable automated actions such as the gathering of forensic timelines across multiple machines, or the isolation of multiple endpoints or even network segments in response to an attack.

Emergent properties

The combination of dynamic communication between endpoints and sharing of metadata and model parameters in the contextual manner described in this document may reveal additional, as yet unimagined mechanisms for further enhancing detection capabilities. Additionally, by running experiments that apply historically collected data over the systems described in this document, we hope to discover previously unpredicted emergent behaviors within some of the participating models. If found, these properties could also form the basis for further research directions.

CONCLUSION

Through experimentation, prototyping, and implementation of new methodologies in our own technology stacks, we have identified a rich set of interactions between models running on endpoints, servers, and the network that have the potential to vastly improve breach detection mechanisms, forensic analysis capabilities, and response capabilities in future cyber security solutions. In this document, we have illustrated some of our earliest findings in this area. Some of the methodologies presented have already been implemented in our solutions. The mechanisms described in this document are likely more widely applicable to our problem space than we currently understand. Continuing research in this area will involve moving more and more logic away from a single central model, and into autonomous, adaptive machine-learning agents that run on, and surround endpoints on the network. By doing this, we expect to be able to utilize emergent behaviors in order to create collective intelligence techniques, based on natural swarm intelligence interactions, where intelligent agents communicate and work together to reach a common goal. This long-term research project will continue under the moniker “Project Blackfin”, and we expect to regularly report new results and findings as they present themselves.

ABOUT F-SECURE

Nobody has better visibility into real-life cyber attacks than F-Secure. We're closing the gap between detection and response, utilizing the unmatched threat intelligence of hundreds of our industry's best technical consultants, millions of devices running our award-winning software, and ceaseless innovations in artificial intelligence. Top banks, airlines, and enterprises trust our commitment to beating the world's most potent threats.

Together with our network of the top channel partners and over 200 service providers, we're on a mission to make sure everyone has the enterprise-grade cyber security we all need. Founded in 1988, F-Secure is listed on the NASDAQ OMX Helsinki Ltd.

f-secure.com | twitter.com/fsecure | linkedin.com/f-secure

F-Secure