OffSecOps

Will Schroeder

@harmj0y

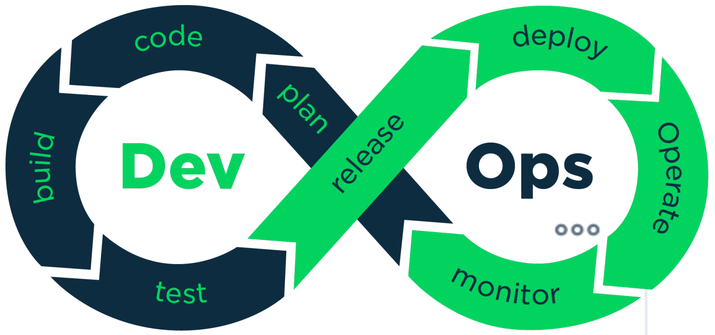
# whoami

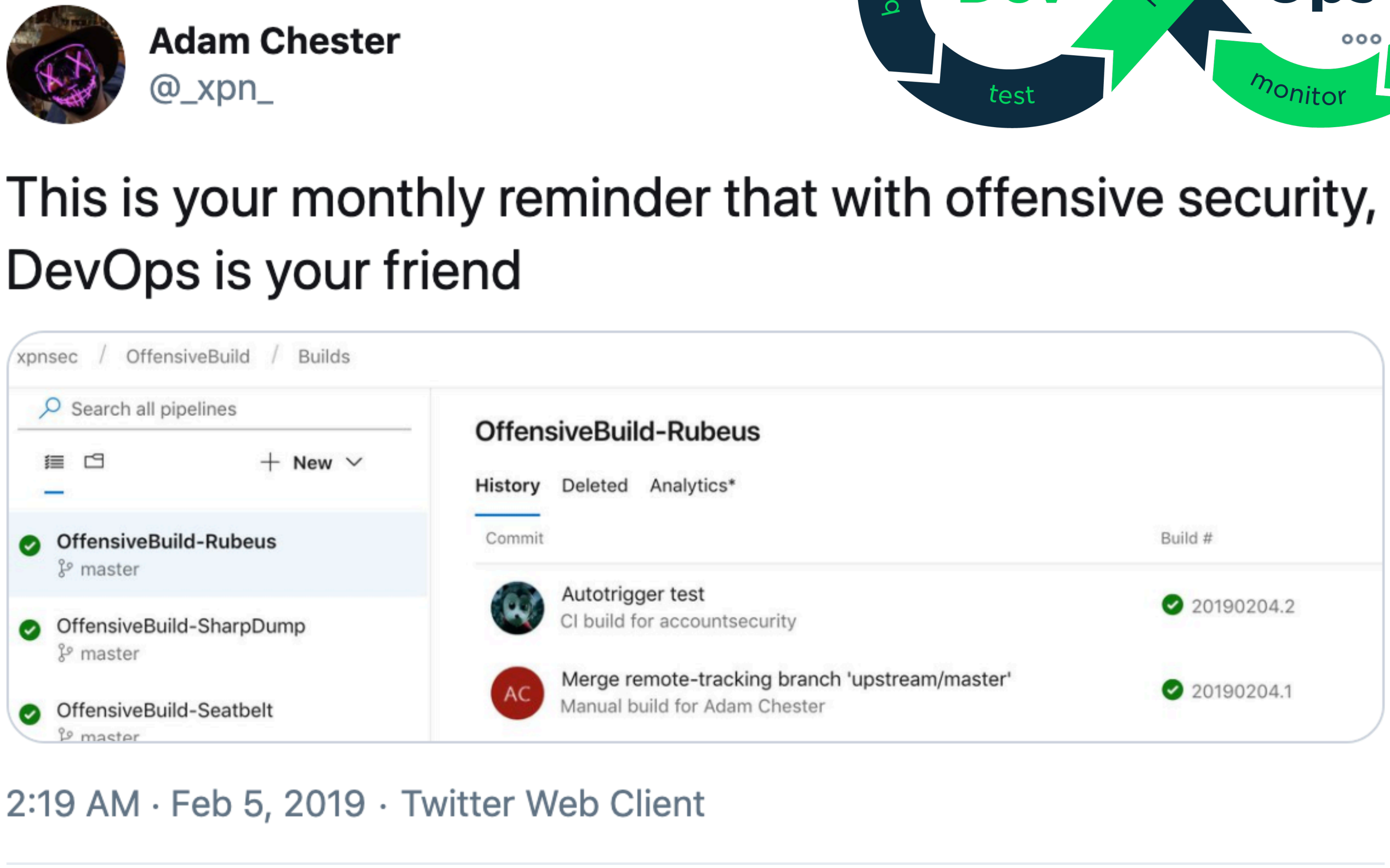
- **Career:** Technical Architect at SpecterOps
- **Code:** Veil-Framework, Empire, PowerView/PowerUp, BloodHound, GhostPack
- **Cons:** DerbyCon (RIP), BlackHat, DEF CON, Troopers, others
- **Content:** Veteran trainer (*Adversary Tactics: Red Team Operations*/others), sometimes blogs at http://blog.harmj0y.net

tl;dr

- Why **#offsec** Needs **#devops**
  - Previous Work
- Our Architecture
  - Declarative Jenkins - structure, library files, parameter passthrough
  - Artifactory - storage, Cobalt Strike integration
- Bonus
  - Proactive checksum scanning, #opsec checks, artifact fingerprinting, etc.

Why #offsec Needs #devops





Adam Chester
@_xpn_

This is your monthly reminder that with offensive security, DevOps is your friend

xpnsec / OffensiveBuild / Builds

Search all pipelines

New

OffensiveBuild-Rubeus
master

OffensiveBuild-SharpDump
master

OffensiveBuild-Seatbelt
master

OffensiveBuild-Rubeus

History Deleted Analytics*

Commit	Build #
Autotrigger test CI build for accountsecurity	20190204.2
Merge remote-tracking branch 'upstream/master' Manual build for Adam Chester	20190204.1

2:19 AM · Feb 5, 2019 · Twitter Web Client

Why **#offsec** Needs **#devops**

- Like any code, offensive tools need testing, proper version control, etc.
- Some special offensive considerations:
  - Obfuscation
  - Indicator stripping
  - Per-op tracking of artifacts
  - Vetting (how much you do *really* trust offensive tool authors? :)
- Lets us transparently insert ourselves into an engagement-critical process and standardize our toolset across all ops

Previous Work

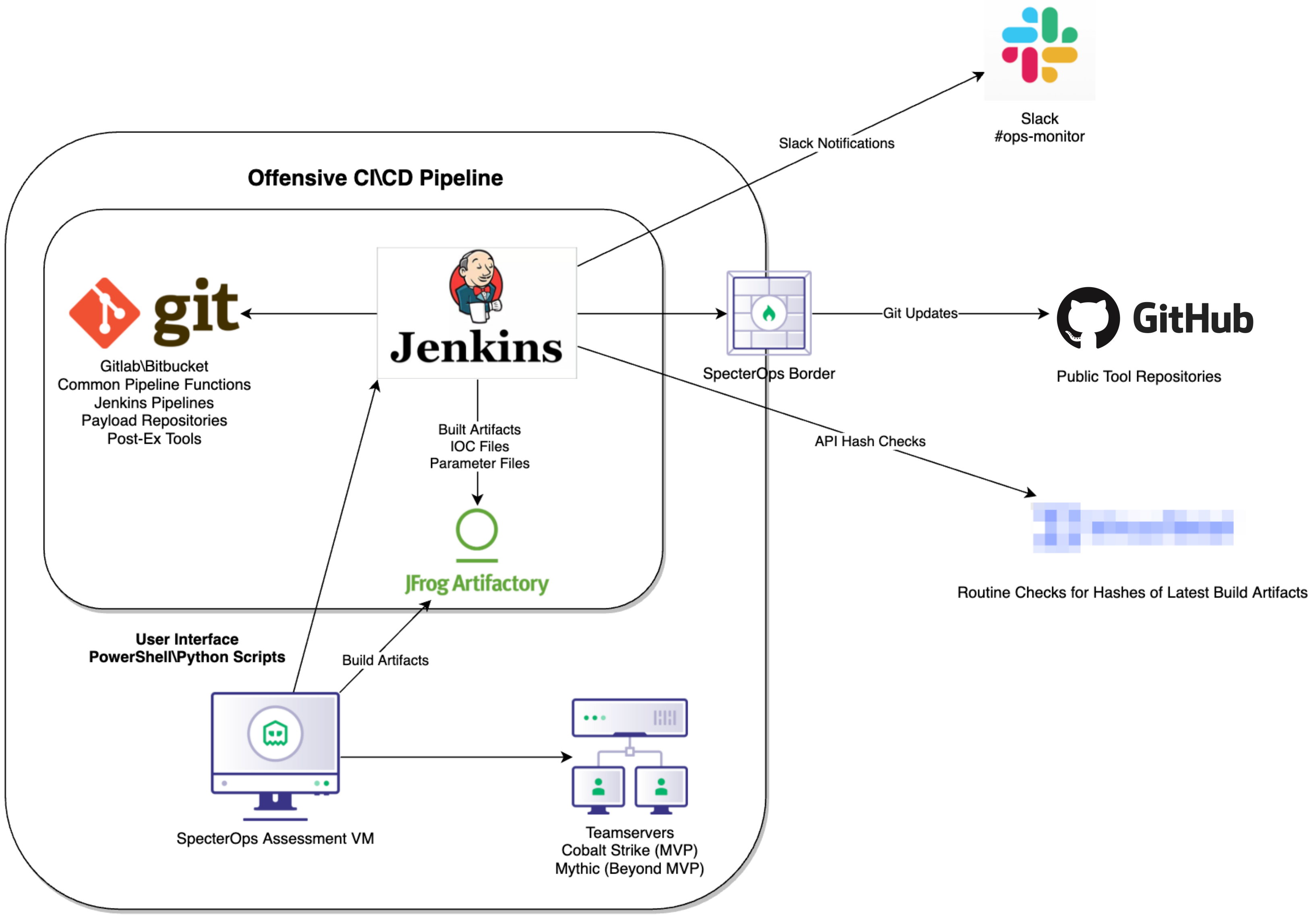
- "*Building, Modifying, and Packing with Azure DevOps*" by @_xpn_
- "*Testing your RedTeam Infrastructure*" by @_xpn_
- "*Offensive Development: How To DevOps Your Red Team*" by @domchell
- Execute-GithubAssembly-Aggressor by MDSec's ActiveBreach Team
- "*Getting Started With Azure DevOps*" by @424f424f
- "*Jenkins - More than Just Target Practice*" by @christruncer

Plenty more that I'm sure I've missed!

Re-inventing The Wheel?

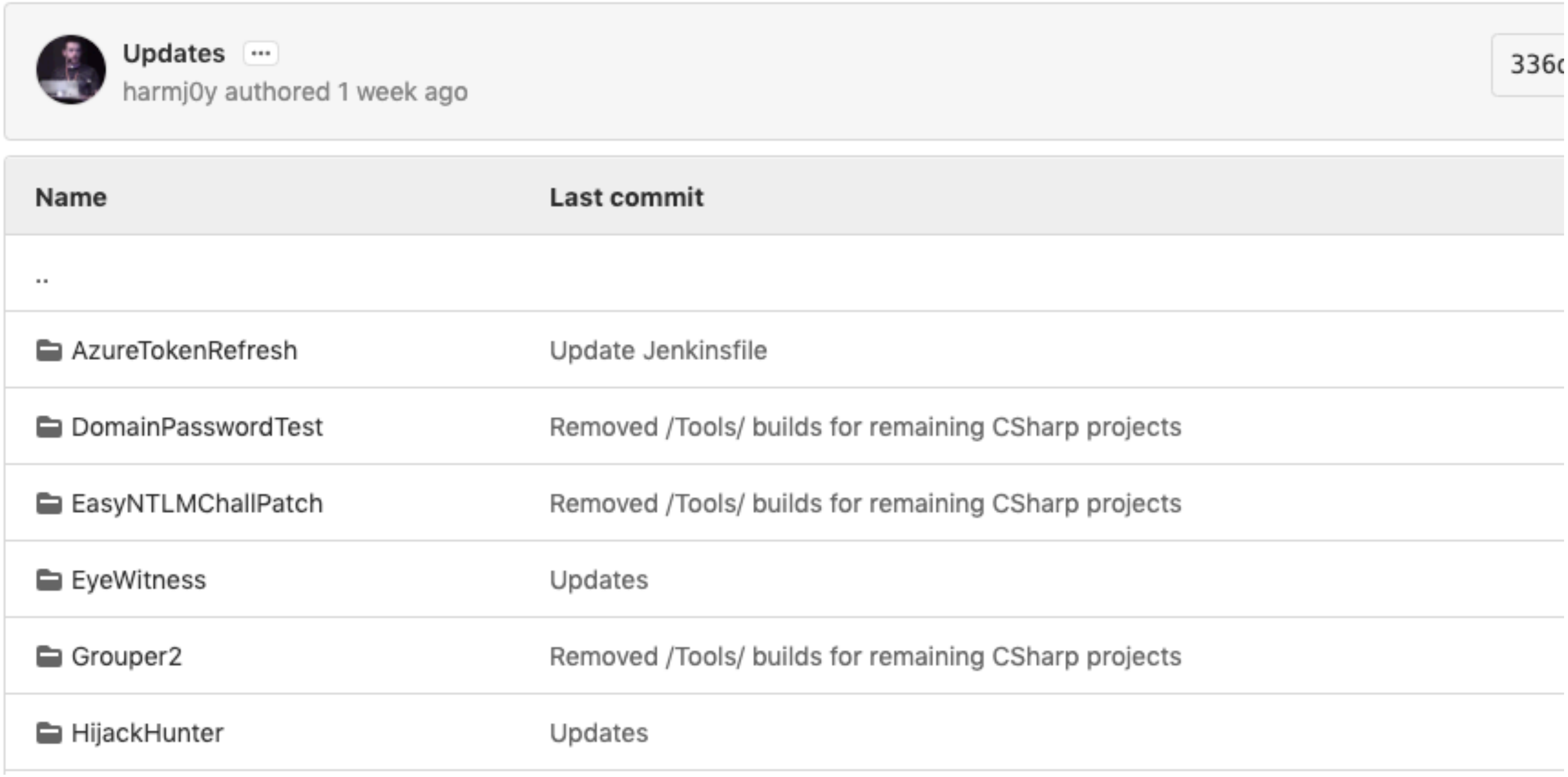
- Why not just use existing AzureDevOps approaches that are out there?
- **Short answer:** we don’t trust ~~Microsoft~~ “organizations” to not collect telemetry on our offensive tools and/or operations
- **Our philosophy**: host as much of your offensive infrastructure (including DevOps pipelines) on hardware that you own and control

Our Architecture



Declarative Jenkins

- Jenkins jobs come in a few flavors:
  - Classic/Freestyle jobs
  - Scripted / **Declarative Pipelines** (the hot new thing™)
- Declarative pipelines allow you to construct your Jenkins builds properly as code
  - We store these on an internal GitLab server in a single repo
- We can also define and use common library functions (more on this shortly)



Updates ···
harmj0y authored 1 week ago

336

Name	Last commit
..	
AzureTokenRefresh	Update Jenkinsfile
DomainPasswordTest	Removed /Tools/ builds for remaining CSharp projects
EasyNTLMChallPatch	Removed /Tools/ builds for remaining CSharp projects
EyeWitness	Updates
Grouper2	Removed /Tools/ builds for remaining CSharp projects
HijackHunter	Updates

```
stage('build') {

    steps {
        // 'ci-jenkins-common' library function -> 'msbuild("PROJECT.sln", ".NET_VERSION")'

        script {
            msbuild("${JOB_NAME}.sln", "3.5")
        }

        script {
            msbuild("${JOB_NAME}.sln", "4.0")
        }
```

```
stage('obfuscation') {
    steps {
        script {
            // obfuscate the binary using a 'ci-jenkins-common' library function
            obfuscateDotnetBinary("3.5")
            obfuscateDotnetBinary("4.0")
        }
    }
```

```
stage('opsec tests') {
    steps {
        script {
            // test the build artifacts using a 'ci-jenkins-common' library function
            testOpsec("${JOB_NAME}\\bin\\3.5\\${JOB_NAME}_3.5.exe")
            testOpsec("${JOB_NAME}\\bin\\3.5\\${JOB_NAME}_3.5_obf.exe")
            testOpsec("${JOB_NAME}\\bin\\4.0\\${JOB_NAME}_4.0.exe")
            testOpsec("${JOB_NAME}\\bin\\4.0\\${JOB_NAME}_4.0_obf.exe")
        }
    }
}
```

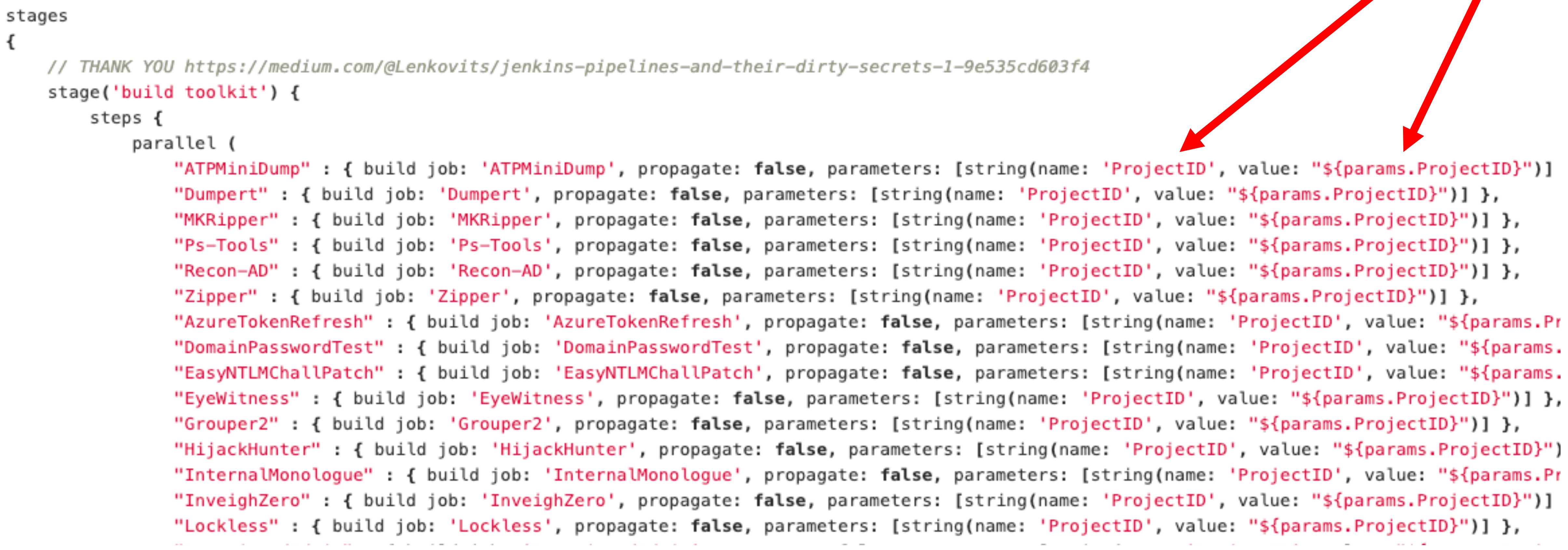
Library Functions

- Jenkins allows for the use of library functions
- These reside in a single internal Gitlab repo, and are cloned down/compiled for any project that references:
  - @Library('ci-jenkins-common') _
- Lets us centralize common functionality and update it across all builds
  - Obfuscation methods/script comment stripping
  - Artifact fingerprint extraction
  - sRDI conversion, etc.

Meta Jobs and Parameter Passthrough

- Declarative Jenkins jobs can take build parameters
- Jenkins jobs can kick off other jobs and pass-through parameters
- This allows us to kick off a build of the entire toolkit with an engagement ID and have that filter down to individual tool builds
    - We also insert a bit of benign randomness into each tool build to ensure uniqueness per engagement/build
- **Translation:** *we can produce unique individual builds for each offensive tool per engagement!*

Meta Jobs and Parameter Passthrough

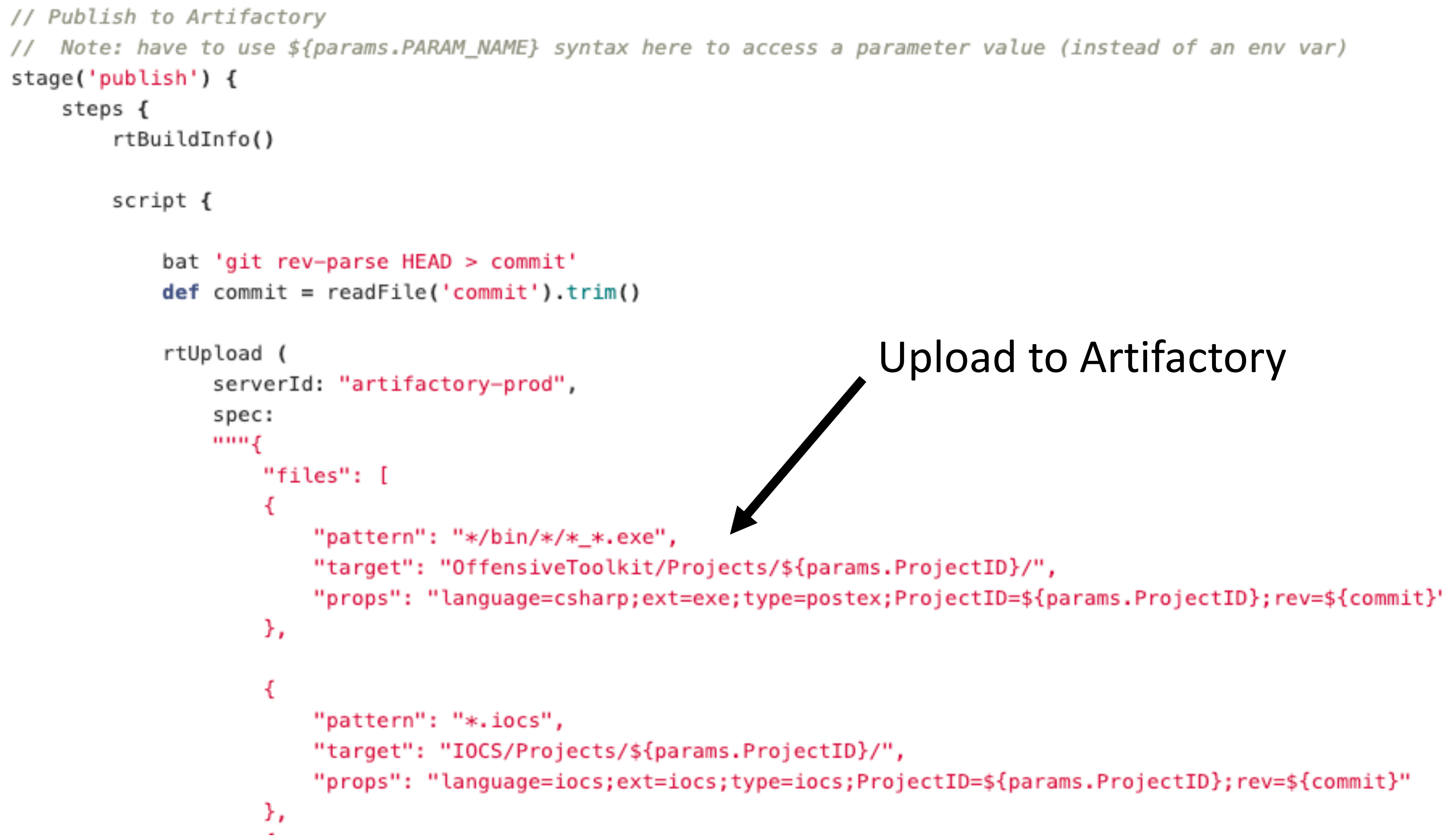


```
stages
{
    // THANK YOU https://medium.com/@Lenkovits/jenkins-pipelines-and-their-dirty-secrets-1-9e535cd603f4
    stage('build toolkit') {
        steps {
            parallel (
                "ATPMiniDump" : { build job: 'ATPMiniDump', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")]
                "Dumpert" : { build job: 'Dumpert', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "MKRipper" : { build job: 'MKRipper', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "Ps-Tools" : { build job: 'Ps-Tools', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "Recon-AD" : { build job: 'Recon-AD', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "Zipper" : { build job: 'Zipper', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "AzureTokenRefresh" : { build job: 'AzureTokenRefresh', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.Pr
                "DomainPasswordTest" : { build job: 'DomainPasswordTest', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.
                "EasyNTLMChallPatch" : { build job: 'EasyNTLMChallPatch', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.
                "EyeWitness" : { build job: 'EyeWitness', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "Grouper2" : { build job: 'Grouper2', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
                "HijackHunter" : { build job: 'HijackHunter', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")
                "InternalMonologue" : { build job: 'InternalMonologue', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.Pr
                "InveighZero" : { build job: 'InveighZero', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")]
                "Lockless" : { build job: 'Lockless', propagate: false, parameters: [string(name: 'ProjectID', value: "${params.ProjectID}")] },
```

Artifactory

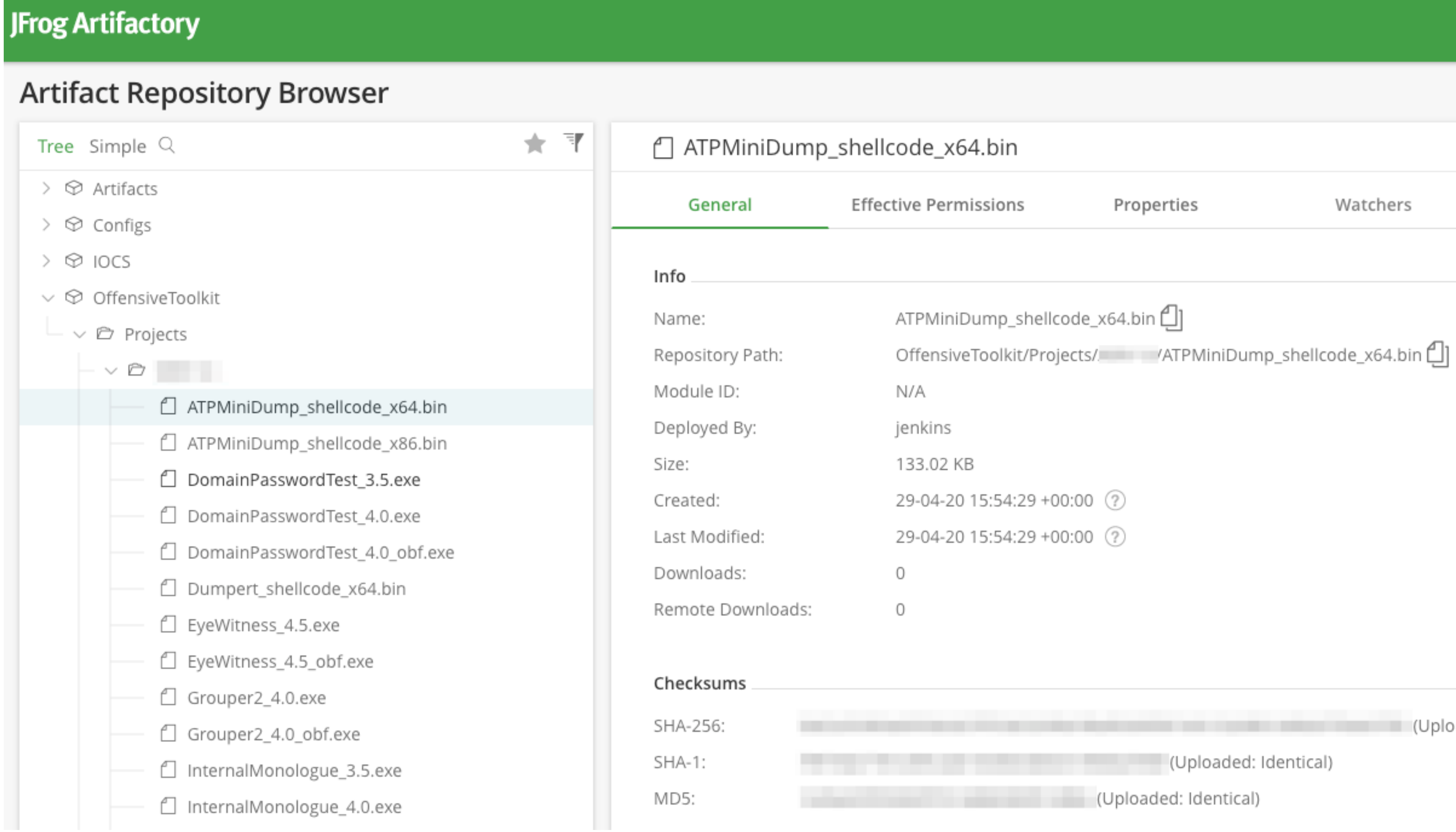
- Functions as the central artifact repository for all of our offensive artifacts (and IOCs!)
  - Artifactory is binary repository – a natural extension to a source code repository, in that it will store the outcome of build processes (aka ‘artifacts’).
- The project ID that’s passed through from the meta build job is used to tag/structure the resulting project folder in Artifactory
- AQL (Artifactory Query Language) and the Artifactory API can be used to query for/retrieve artifacts for a specific engagement

Jenkins + Artifactory



```
// Publish to Artifactory
//  Note: have to use ${params.PARAM_NAME} syntax here to access a parameter value (instead of an env var)
stage('publish') {
    steps {
        rtBuildInfo()

        script {

            bat 'git rev-parse HEAD > commit'
            def commit = readFile('commit').trim()

            rtUpload (
                serverId: "artifactory-prod",
                spec:
                """{
                    "files": [
                    {
                        "pattern": "*/bin/*/*_*.exe",
                        "target": "OffensiveToolkit/Projects/${params.ProjectID}/",
                        "props": "language=csharp;ext=exe;type=postex;ProjectID=${params.ProjectID};rev=${commit}"
                    },

                    {
                        "pattern": "*.iocs",
                        "target": "IOCS/Projects/${params.ProjectID}/",
                        "props": "language=iocs;ext=iocs;type=iocs;ProjectID=${params.ProjectID};rev=${commit}"
                    },
```

Upload to Artifactory


JFrog Artifactory

Artifact Repository Browser

Tree Simple

- Artifacts
- Configs
- IOCS
- OffensiveToolkit
  - Projects
    - [redacted]
      - ATPMiniDump_shellcode_x64.bin
      - ATPMiniDump_shellcode_x86.bin
      - DomainPasswordTest_3.5.exe
      - DomainPasswordTest_4.0.exe
      - DomainPasswordTest_4.0_obf.exe
      - Dumpert_shellcode_x64.bin
      - EyeWitness_4.5.exe
      - EyeWitness_4.5_obf.exe
      - Grouper2_4.0.exe
      - Grouper2_4.0_obf.exe
      - InternalMonologue_3.5.exe
      - InternalMonologue_4.0.exe

ATPMiniDump_shellcode_x64.bin

General | Effective Permissions | Properties | Watchers

Info

Name:	ATPMiniDump_shellcode_x64.bin
Repository Path:	OffensiveToolkit/Projects/[redacted]/ATPMiniDump_shellcode_x64.bin
Module ID:	N/A
Deployed By:	jenkins
Size:	133.02 KB
Created:	29-04-20 15:54:29 +00:00
Last Modified:	29-04-20 15:54:29 +00:00
Downloads:	0
Remote Downloads:	0

Checksums

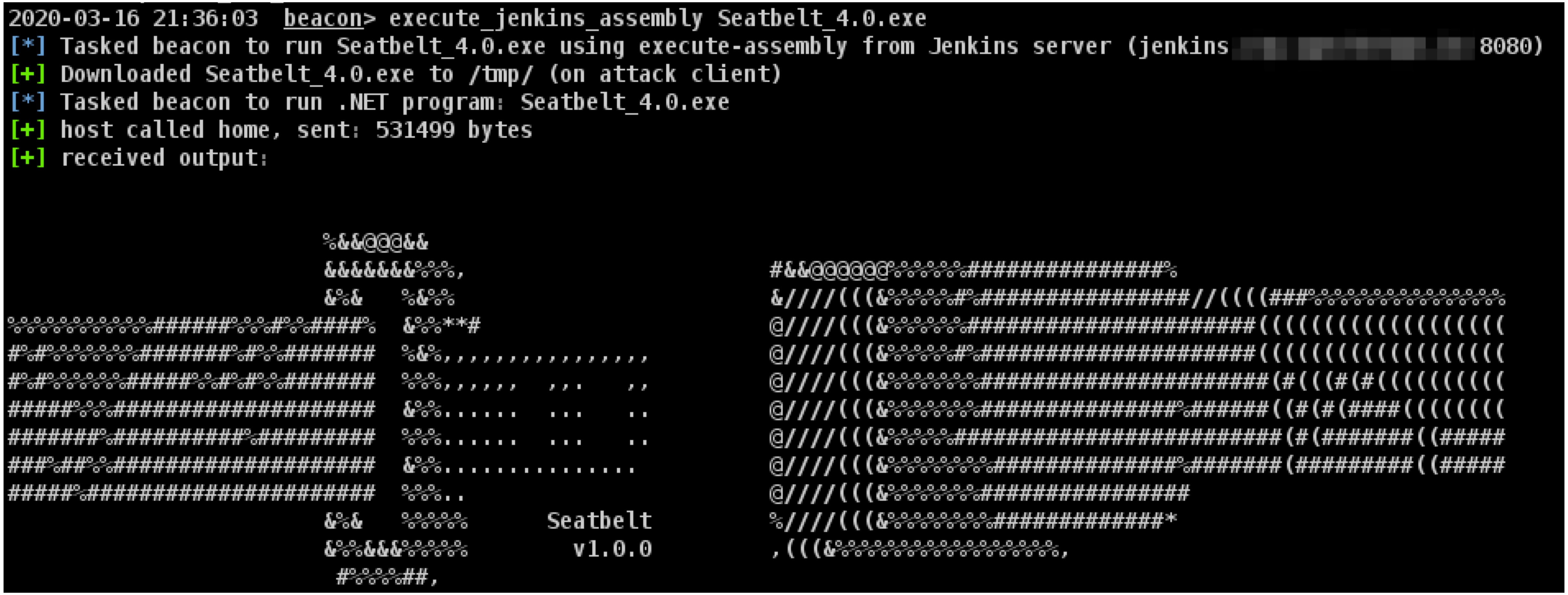
SHA-256:	[redacted] (Uplo
SHA-1:	[redacted] (Uploaded: Identical)
MD5:	[redacted] (Uploaded: Identical)

Artifactory: Operator Interfaces

- We adapted MDSec's Execute-GithubAssembly-Aggressor to pull artifacts from our local Artifactory instance, for the specified engagement ID for the op
    - Allows operators to easily run any C#, PIC, or PowerShell payload through a native Beacon

- We also have manual PowerShell/Python scripts that interact with the Jenkins "API" and Artifactory API to build and retrieve artifacts

Artifactory + Cobalt Strike



```
2020-03-16 21:36:03  beacon> execute_jenkins_assembly Seatbelt_4.0.exe
[*] Tasked beacon to run Seatbelt_4.0.exe using execute-assembly from Jenkins server (jenkins[redacted]8080)
[+] Downloaded Seatbelt_4.0.exe to /tmp/ (on attack client)
[*] Tasked beacon to run .NET program: Seatbelt_4.0.exe
[+] host called home, sent: 531499 bytes
[+] received output:


                        %&&@@@&&
                        &&&&&&&%%%,                       #&&@@@@@@%%%%%%###############%
                        &%&   %&%%                        &////(((&%%%%%#%################//((((###%%%%%%%%%%%%%%%
%%%%%%%%%%%######%%%#%%####%  &%%**#                      @////(((&%%%%%%######################(((((((((((((((((((
#%#%%%%%%%#######%#%%#######  %&%,,,,,,,,,,,,,,,,         @////(((&%%%%%#%#####################(((((((((((((((((((
#%#%%%%%%#####%%#%#%########  %%%,,,,,,  ,,.   ,,         @////(((&%%%%%%%######################(#(((#(#((((((((((
#####%%%####################  &%%......  ...   ..         @////(((&%%%%%%%###############%######((#(#(####((((((((
#######%##########%#########  %%%......  ...   ..         @////(((&%%%%%#########################(#(#######((#####
###%##%%####################  &%%...............          @////(((&%%%%%%%%##############%#######(#########((#####
#####%######################  %%%..                       @////(((&%%%%%%%################
                        &%&   %%%%%      Seatbelt         %////(((&%%%%%%%%#############*
                        &%%&&&%%%%%        v1.0.0         ,(((&%%%%%%%%%%%%%%%%%,
                         #%%%%##,
```

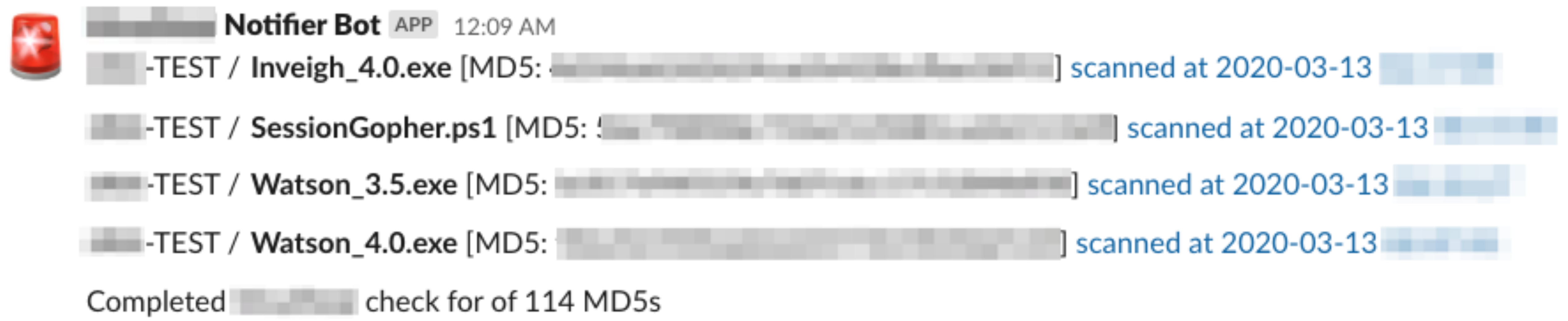
Now that we have our Offensive Toolkit defined as code, what else can we do with this architecture?

Submitted Artifact Detection

- Remember, each artifact is unique per engagement, and has engagement ID passed through as metadata

- Why don't we build another Jenkins Declarative Pipeline that periodically:
  - Pulls checksums from Artifactory for currently built + used tools
  - Scans an online submission service to see if any artifacts from recent engagements were submitted, alerting in Slack with the tool name + ID

Submitted Artifact Detection



[redacted] Notifier Bot APP 12:09 AM

[redacted]-TEST / **Inveigh_4.0.exe** [MD5: [redacted]] scanned at 2020-03-13 [redacted]

[redacted]-TEST / **SessionGopher.ps1** [MD5: [redacted]] scanned at 2020-03-13 [redacted]

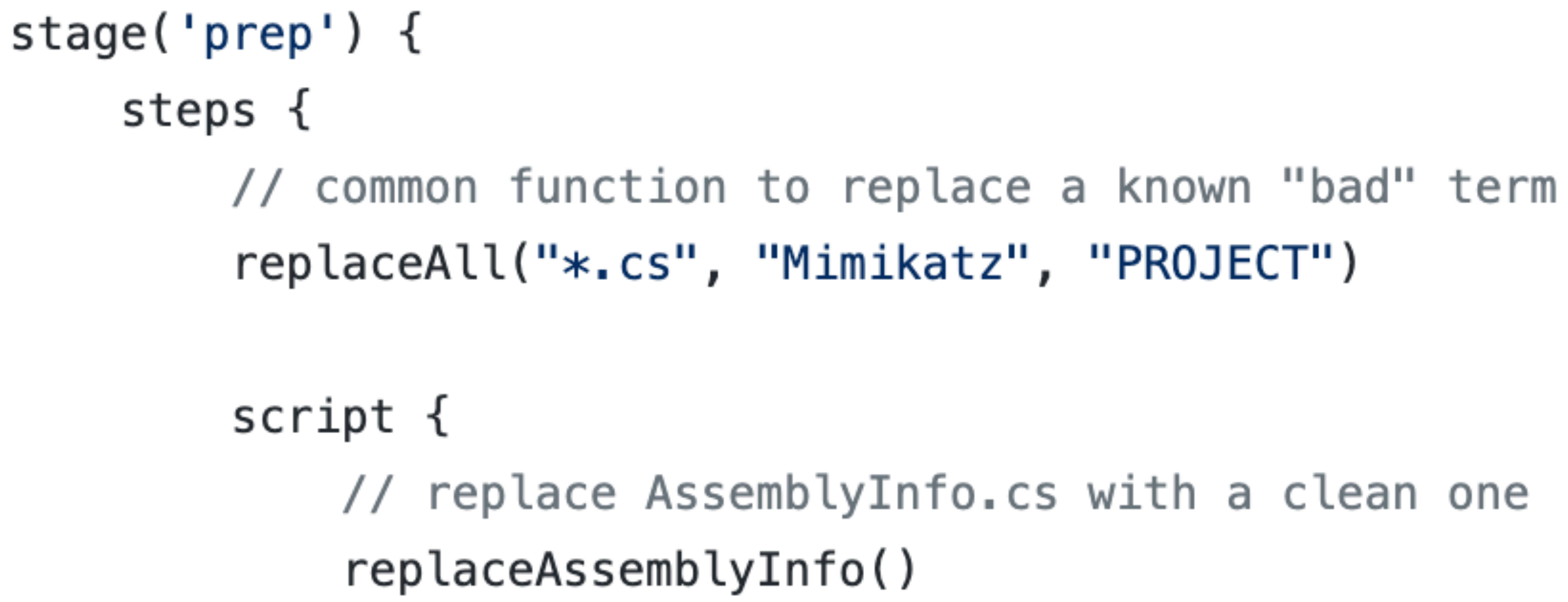
[redacted]-TEST / **Watson_3.5.exe** [MD5: [redacted]] scanned at 2020-03-13 [redacted]

[redacted]-TEST / **Watson_4.0.exe** [MD5: [redacted]] scanned at 2020-03-13 [redacted]

Completed [redacted] check for of 114 MD5s

Indicator Stripping

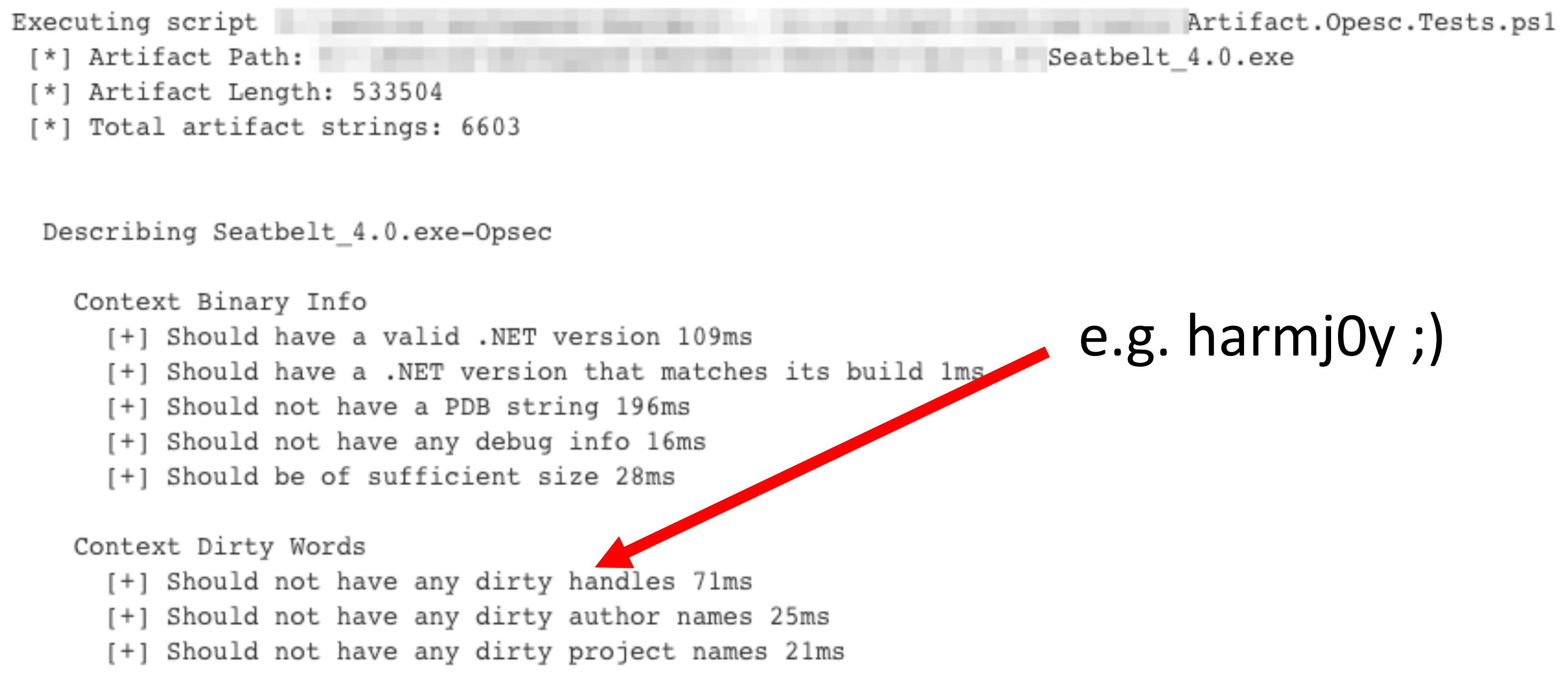
- If tools have any known IOCs/“dirty” terms (even if the code is public and we don’t control it) we can do preprocessing on the build side



```
stage('prep') {
    steps {
        // common function to replace a known "bad" term
        replaceAll("*.cs", "Mimikatz", "PROJECT")

        script {
            // replace AssemblyInfo.cs with a clean one
            replaceAssemblyInfo()
```

#opsec Checks

- Since we've hooked ourselves into the tool build process, we can proactively scan every tool build for simple #opsec fails

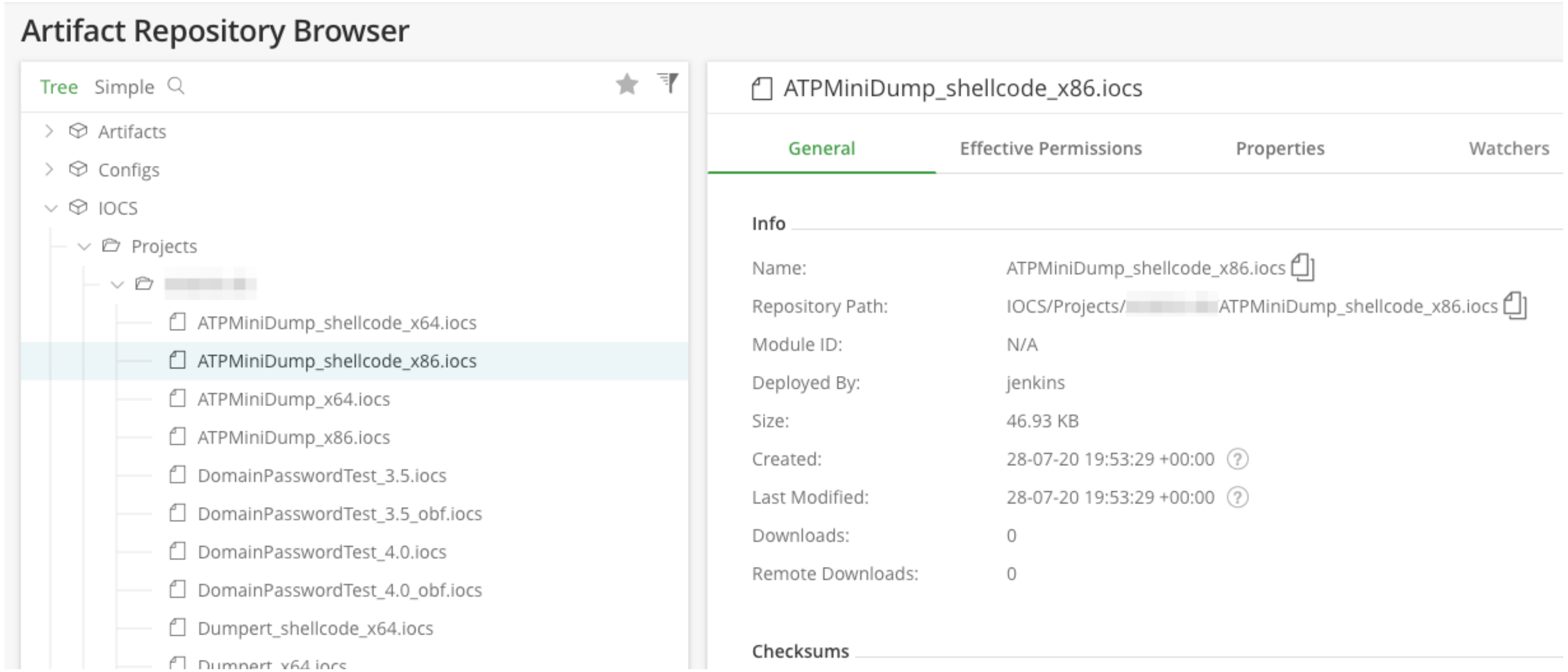


```
Executing script [redacted]Artifact.Opesc.Tests.ps1
 [*] Artifact Path: [redacted]Seatbelt_4.0.exe
 [*] Artifact Length: 533504
 [*] Total artifact strings: 6603


  Describing Seatbelt_4.0.exe-Opsec

    Context Binary Info
      [+] Should have a valid .NET version 109ms
      [+] Should have a .NET version that matches its build 1ms
      [+] Should not have a PDB string 196ms
      [+] Should not have any debug info 16ms
      [+] Should be of sufficient size 28ms

    Context Dirty Words
      [+] Should not have any dirty handles 71ms
      [+] Should not have any dirty author names 25ms
      [+] Should not have any dirty project names 21ms
```

e.g. harmj0y ;)

Artifact Fingerprinting

- We can also do custom fingerprinting of every artifact generated



Artifact Repository Browser

Tree Simple

- Artifacts
- Configs
- IOCS
  - Projects
    - [redacted]
      - ATPMiniDump_shellcode_x64.iocs
      - ATPMiniDump_shellcode_x86.iocs
      - ATPMiniDump_x64.iocs
      - ATPMiniDump_x86.iocs
      - DomainPasswordTest_3.5.iocs
      - DomainPasswordTest_3.5_obf.iocs
      - DomainPasswordTest_4.0.iocs
      - DomainPasswordTest_4.0_obf.iocs
      - Dumpert_shellcode_x64.iocs
      - Dumpert_x64.iocs

ATPMiniDump_shellcode_x86.iocs

General | Effective Permissions | Properties | Watchers

Info

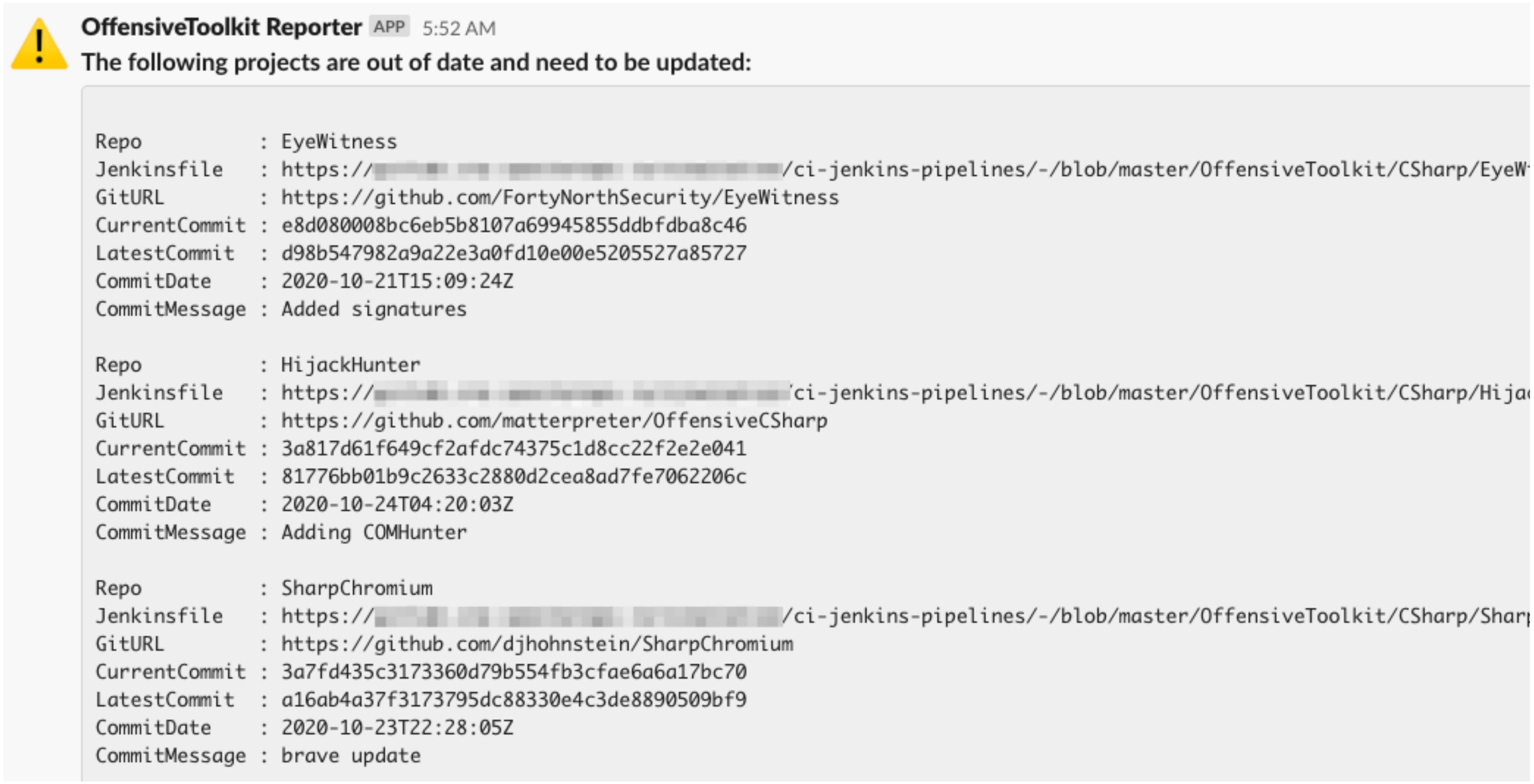
Name:	ATPMiniDump_shellcode_x86.iocs
Repository Path:	IOCS/Projects/[redacted]ATPMiniDump_shellcode_x86.iocs
Module ID:	N/A
Deployed By:	jenkins
Size:	46.93 KB
Created:	28-07-20 19:53:29 +00:00
Last Modified:	28-07-20 19:53:29 +00:00
Downloads:	0
Remote Downloads:	0

Checksums

Project Vetting

- For projects we don't control, we don't want to just pull in master branches of any update immediately
  - We're deploying this code on sensitive customer systems!
- Our update process:
  - All OffensiveToolkit Jenkinsfiles are tagged to a specific SHA1 commit
  - Every Monday, a Jenkins build job clones down the OffensiveToolkit repo
  - The commit in the file is compared to the most recent project commit
  - Deltas are reported to Slack, then manually reviewed/updated

How much do *you* trust offensive developers? ;)



OffensiveToolkit Reporter APP 5:52 AM

The following projects are out of date and need to be updated:

```
Repo          : EyeWitness
Jenkinsfile   : https://[redacted]/ci-jenkins-pipelines/-/blob/master/OffensiveToolkit/CSharp/EyeW
GitURL        : https://github.com/FortyNorthSecurity/EyeWitness
CurrentCommit : e8d080008bc6eb5b8107a69945855ddbfdba8c46
LatestCommit  : d98b547982a9a22e3a0fd10e00e5205527a85727
CommitDate    : 2020-10-21T15:09:24Z
CommitMessage : Added signatures

Repo          : HijackHunter
Jenkinsfile   : https://[redacted]ci-jenkins-pipelines/-/blob/master/OffensiveToolkit/CSharp/Hija
GitURL        : https://github.com/matterpreter/OffensiveCSharp
CurrentCommit : 3a817d61f649cf2afdc74375c1d8cc22f2e2e041
LatestCommit  : 81776bb01b9c2633c2880d2cea8ad7fe7062206c
CommitDate    : 2020-10-24T04:20:03Z
CommitMessage : Adding COMHunter

Repo          : SharpChromium
Jenkinsfile   : https://[redacted]/ci-jenkins-pipelines/-/blob/master/OffensiveToolkit/CSharp/Shar
GitURL        : https://github.com/djhohnstein/SharpChromium
CurrentCommit : 3a7fd435c3173360d79b554fb3cfae6a6a17bc70
LatestCommit  : a16ab4a37f3173795dc88330e4c3de8890509bf9
CommitDate    : 2020-10-23T22:28:05Z
CommitMessage : brave update
```

Wrapup

- Consider integrating DevOps practices into your offensive operations!
  - You can build all of these on your own hardware with SCM + Jenkins + Artifactory + a bit of sweat
- Hooking into the tool build/generation/deployment process allows for various creative applications across operations
- (Current dev) These approaches can also apply for payload generation
  - Opsec checks, proactive checksum scanning, detonation for testing, etc.

Thanks!

- Any questions?
- Sample Declarative Pipeline (Poc||GTFO): https://bit.ly/324V8Au
- @harmj0y on Twitter and the BloodHound Slack
- will [at] harmj0y.net

www.specterops.io
@specterops
info@specterops.io