# 599.3
# Preventing Exploitation

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

SEC599.3 | Defeating Advanced Adversaries

SANS

# Preventing Exploitation



This page intentionally left blank.

## TABLE OF CONTENTS





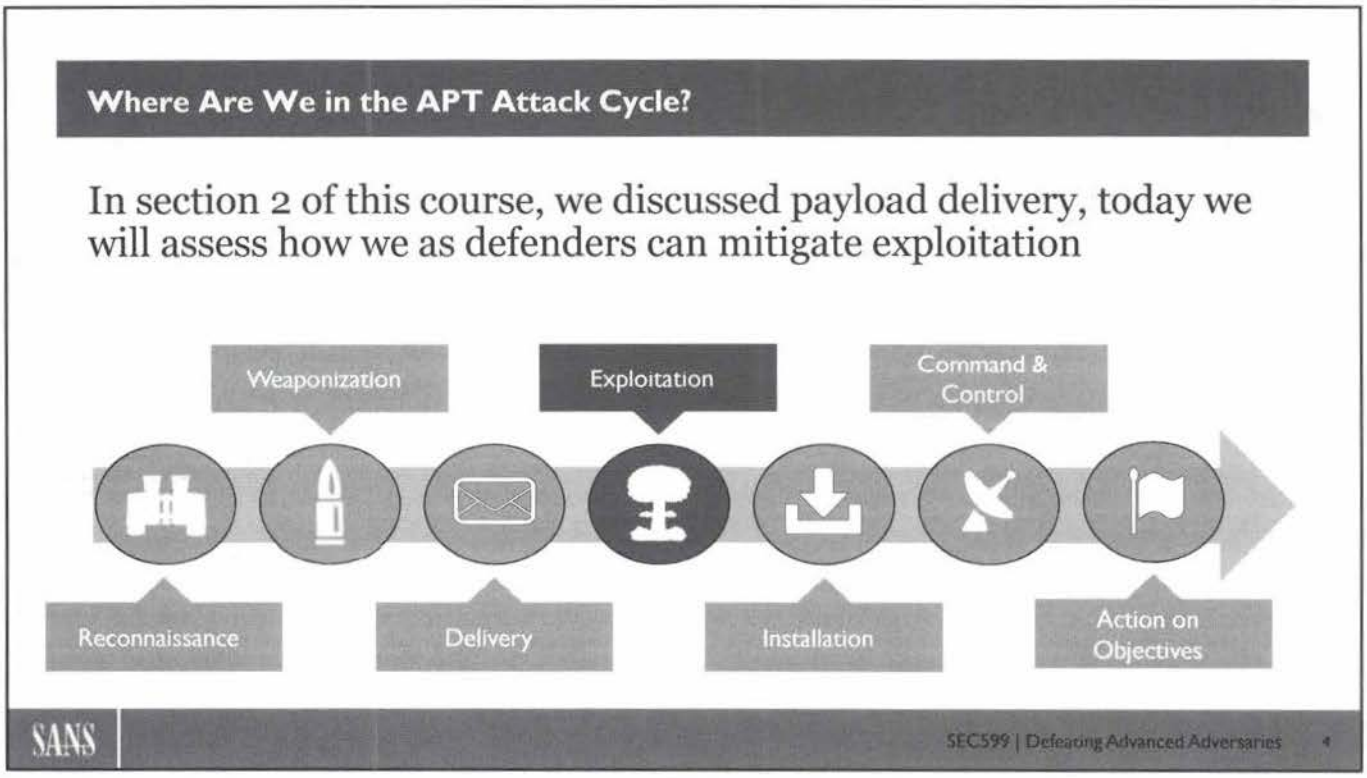
This page intentionally left blank.

## TABLE OF CONTENTS





This page intentionally left blank.




**Where Are We in the APT Attack Cycle?**

Welcome to Day 3 of SEC599! Yesterday, we saw how adversaries could deliver their weaponized payloads during the delivery phase of the APT Attack Cycle. We will now assess how exploitation of vulnerabilities by the payload can be prevented or detected.

*Adversary perspective*

After the delivery, the malware will be executed on devices used by the victims. When a user receives a payload, and opens / runs it, it will run with the privileges of that user. Depending on the goals of the adversaries, running the executable with the user's privileges might be sufficient or not (we assume that the target user is a least privilege user, e.g. not a Windows administrator). If higher privileges are needed, the adversaries will need to use exploits to gain privilege escalation. This can be as simple as a User Account Control bypass, or as sophisticated as a zero-day to achieve code execution in the Windows kernel.

Adversaries often use a combination of technical and people exploits. Consider the example of malicious macro in an office document: it will not run as long as the human doesn't open the document (and enables macros, if required). Next to generating the malicious payload, the adversary will still need to convince the target to open / run it.

*Defender perspective*

As defenders, we have two opportunities during this step to combat adversaries: prevent exploitation and detect exploitation.

As part of vulnerability management, we should:

- Harden our systems according to a baseline configuration (shut down unneeded services, change default credentials…). NIST is well-known for producing quality hardening guidelines;
- Ensure patches for third-party software are timely installed;
- Ensure the software we develop ourselves is developed according to secure development standards;

- Ensure our entire IT environment (including infrastructure and applications) is regularly assessed for vulnerabilities (and that identified flaws are fixed);
- Have a formal vulnerability management process in place, where vulnerabilities have to be formally accepted if they are not mitigated.

Application whitelisting can be of great help to prevent arbitrary code execution, but it requires a standardized software environment. If users are expected to install and run their own software according to their business needs, then application whitelisting will be extremely hard to successfully implement. Windows provides free-to-use application whitelisting technology via AppLocker and Software Restriction Policies.

Another interesting (though end-of-life) technology developed by Windows is EMET (Enhanced Mitigation Experience Tool), which attempts to protect endpoints from successful exploitation. Its features are implemented (in part) in recent Windows Operating Systems such as Windows 10 Enterprise.

Monitoring all code execution on a system is not only a great way to detect exploitation, but it also provides logs that can be used in forensic investigations of successful attacks. Microsoft's Sysmon is a free tool that collects information of code execution and writes it to a Windows event log. For example, Sysmon can create Windows event log entries for executed programs, including a cryptographic hash of the executable image.

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

**Securing the Network**
- Network Access Control & 802.1X

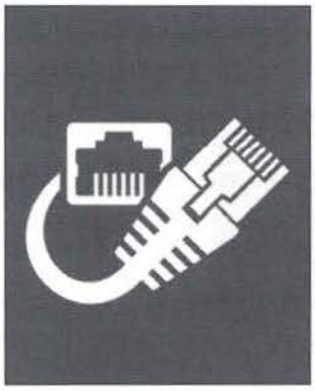
**Securing Software**
- Software Development Lifecycle (SDL) & Threat Modeling
- Vulnerability Assessments
- Exercise: Authenticated Scans Using Nessus
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation Using Compile-Time Controls
- Exploit Mitigation Techniques – Exploit Guard, EMET, & Others
- Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**
- OS Hardening & Best Practices
- Endpoint Protection Solutions
- Application Whitelisting to Stop Payload Execution
- Exercise: Configuring AppLocker



This page intentionally left blank.

## Network Access Control & 802.1X



The problem: Ethernet networks are open. If you can plug your laptop into an Ethernet socket, you will get network access.

Your machine will be assigned an IP address via DHCP, or you will configure it with a static IP address.

From then on, your machine will be able to use the IP protocol on the network. No identification, authentication or authorization is required.



**Network Access Control & 802.1X**

Ethernet networks were designed to be open. If you have physical access to an Ethernet socket, you can get access to the network (we are considering networks without Network Access Control).

When you plug in the Ethernet cable of your laptop into an Ethernet socket, the Ethernet card on your laptop and the network device (typically a switch) will exchange signals over the Ethernet cable to establish an Ethernet connection.

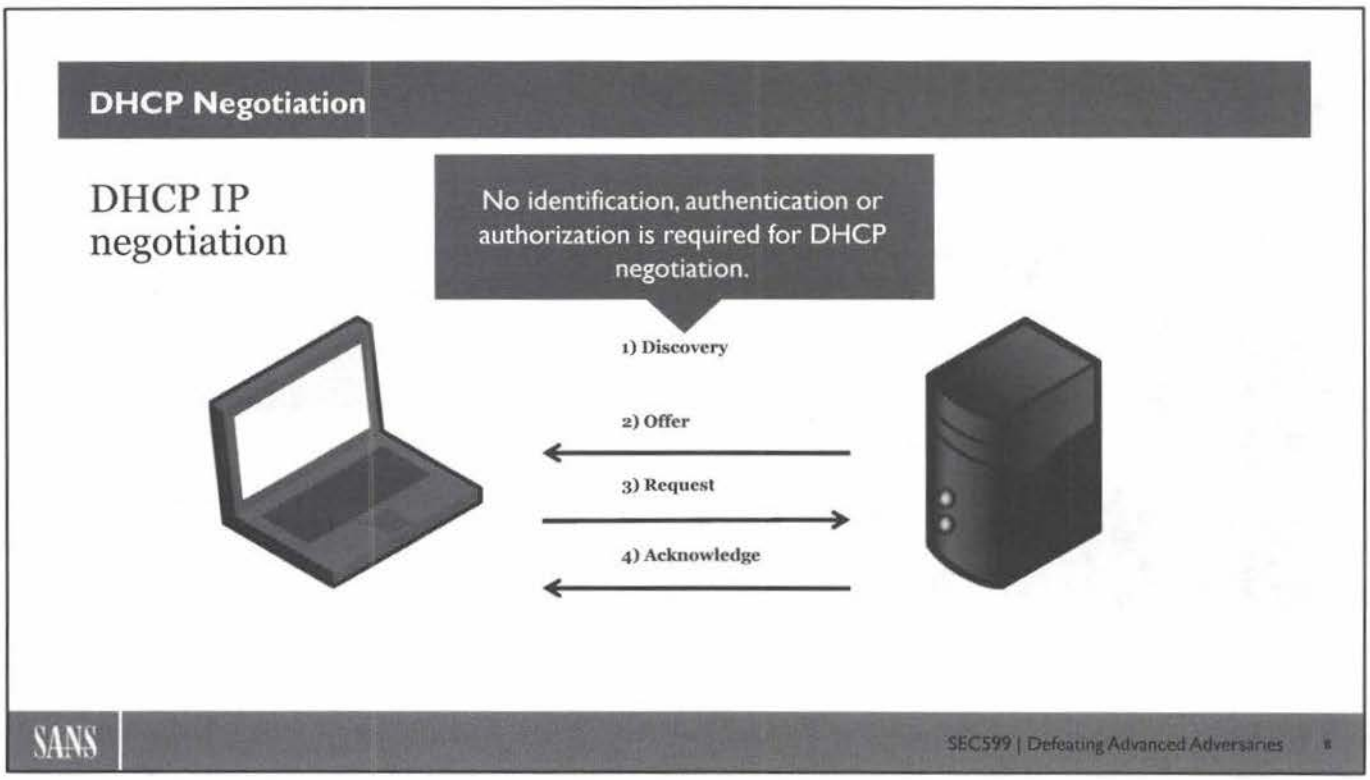
Once you have an Ethernet connection, your laptop can initiate IP traffic. To send and receive IP packets, your laptop needs an IP address (IPv4 or IPv6 or both). In the majority of networks, your laptop will receive an IP address from a DHCP server. The Dynamic Host Configuration Protocol is a protocol used to automatically configure the host (your laptop) with properties it needs to operate in the network. An IP address is an essential property set through DHCP.

If DHCP is not available, you can configure your laptop with a static IP address (this might require some reconnaissance to figure out with IP range to use).

From then on, without identification, authentication or authorization, your laptop will be able to use the IP protocol on the network (and thus TCP, UDP…). Remark that this network access does not imply unauthorized access to all resources on the network: on many networks, Active Directory is implemented to secure resources. But you will be able for example to conduct port scans, attempt to sniff traffic.

So, physical access means network access. And this is a problem for enterprises: they have so many premises that it is very hard to physically secure all network access points.

Remark that with Wi-Fi we are dealing with a shared spectrum by design, so we'll have to place more focus on logical controls (authentication, encryption…)




**DHCP Negotiation**

To obtain an IP address (and other properties, like gateway IP address, DNS server IP address…) when connecting to a network, a machine will initiate a DHCP communication.

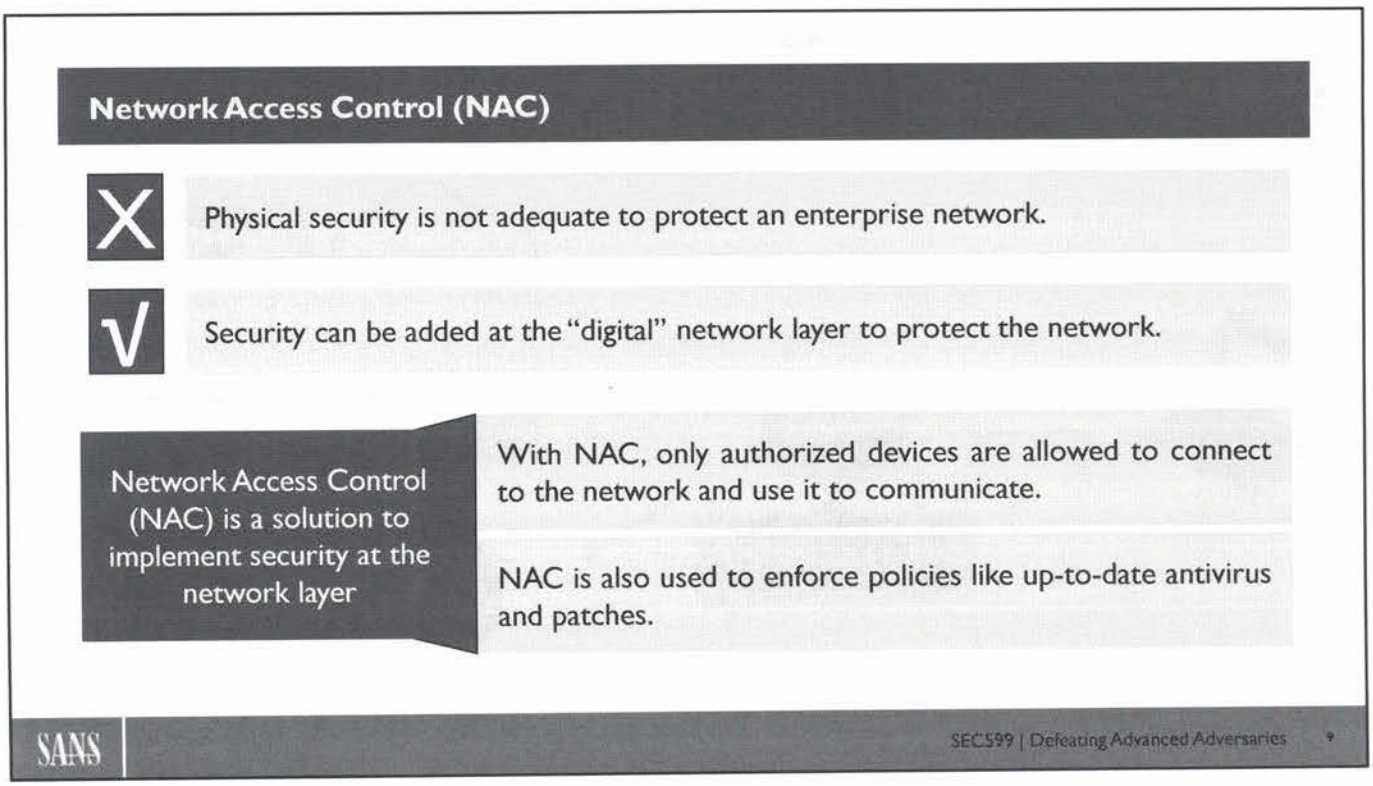
First, it will send out a broadcast Discovery message (UDP packets).

DHCP servers on the network will listen to Discovery messages broadcasted on the network, and reply to the client with an Offer message. This offer message will contain the clients MAC address and a proposed IP address.

The client will select one DHCP server to continue the negotiation, and broadcast a Request message to that server accepting the proposed IP address.

Finally, the DHCP server will acknowledge the request.

With this information, the client can configure its network interface. This does not require identification, authentication or authorization.




**Network Access Control (NAC)**
Since (large) enterprises cannot rely on physical security to protect their network, other solutions had to be found.

One solution is to add security at the "digital" network layer: before a client is allowed full access to the network, it needs to identify and authenticate itself.

Only when authentication succeeds, the client will be allowed to access the network. Failing to authenticate, the client will not be able to send or receive network traffic.

Network Access Control (NAC) is a solution to achieve this level of security. With NAC, only authorized devices can use the corporate network. The administrator of the network needs to authorize devices, and he / she can deauthorize devices (for example in case of laptop theft). Without authorization, devices are not allowed to connect to the network.

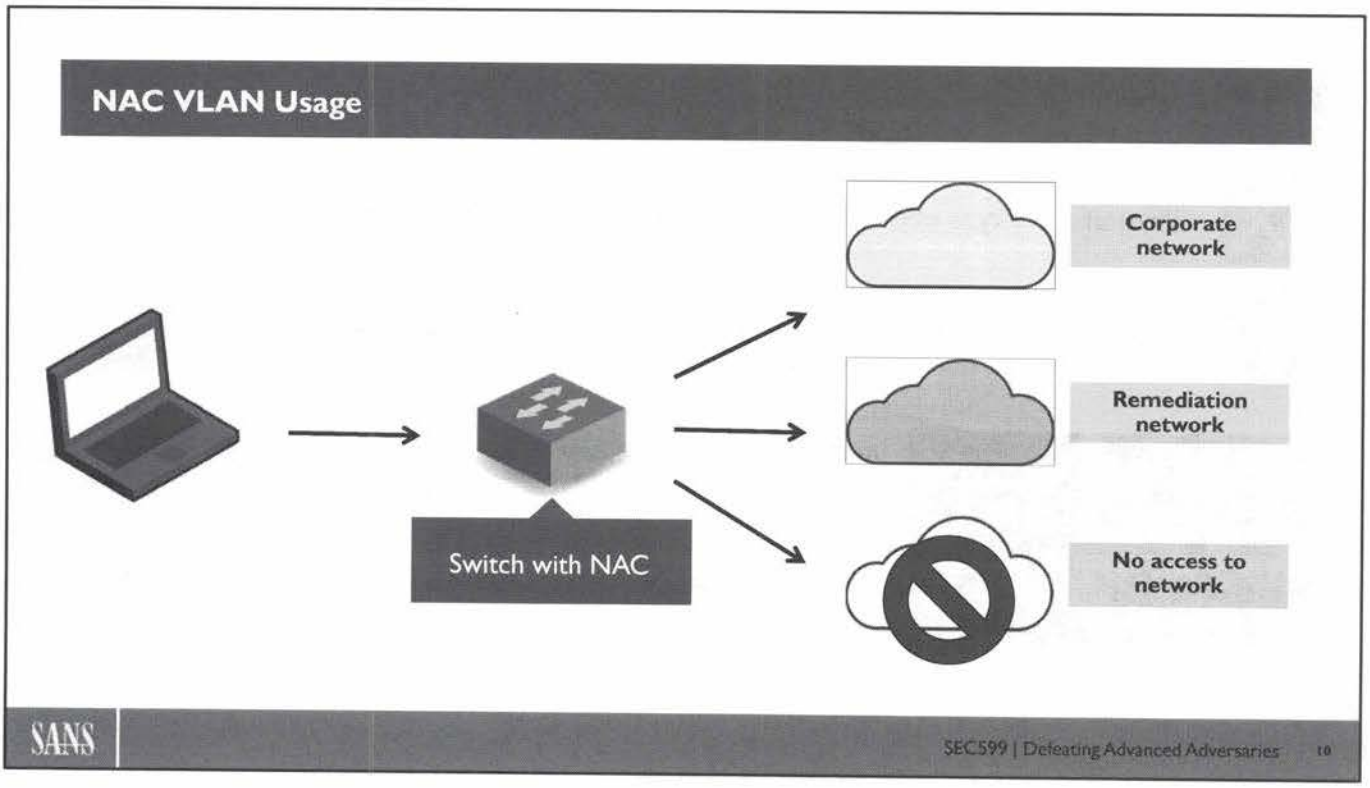
Implementing NAC requires a substantial amount of resources (client devices and network devices need to support NAC, NAC needs to be configured and maintained…) and because of this cost, other applications of NAC were developed.

For example, with NAC, it is also possible to allow or deny access to the network for a client based on the properties of the client. With NAC, it is possible to inspect the client, and only allow access if the client has a working antivirus with up-to-date signatures and if the latest patches were applied.

This requires a NAC agent running on the operating system to inspect the machine.

Often, clients who are not compliant with these policies are given access to a remediation network instead of being blocked from network access.

In this remediation network, resources are available to update the client to make it compliant with the policies.




**NAC VLAN Usage**
NAC is often implemented according to the diagram shown above.

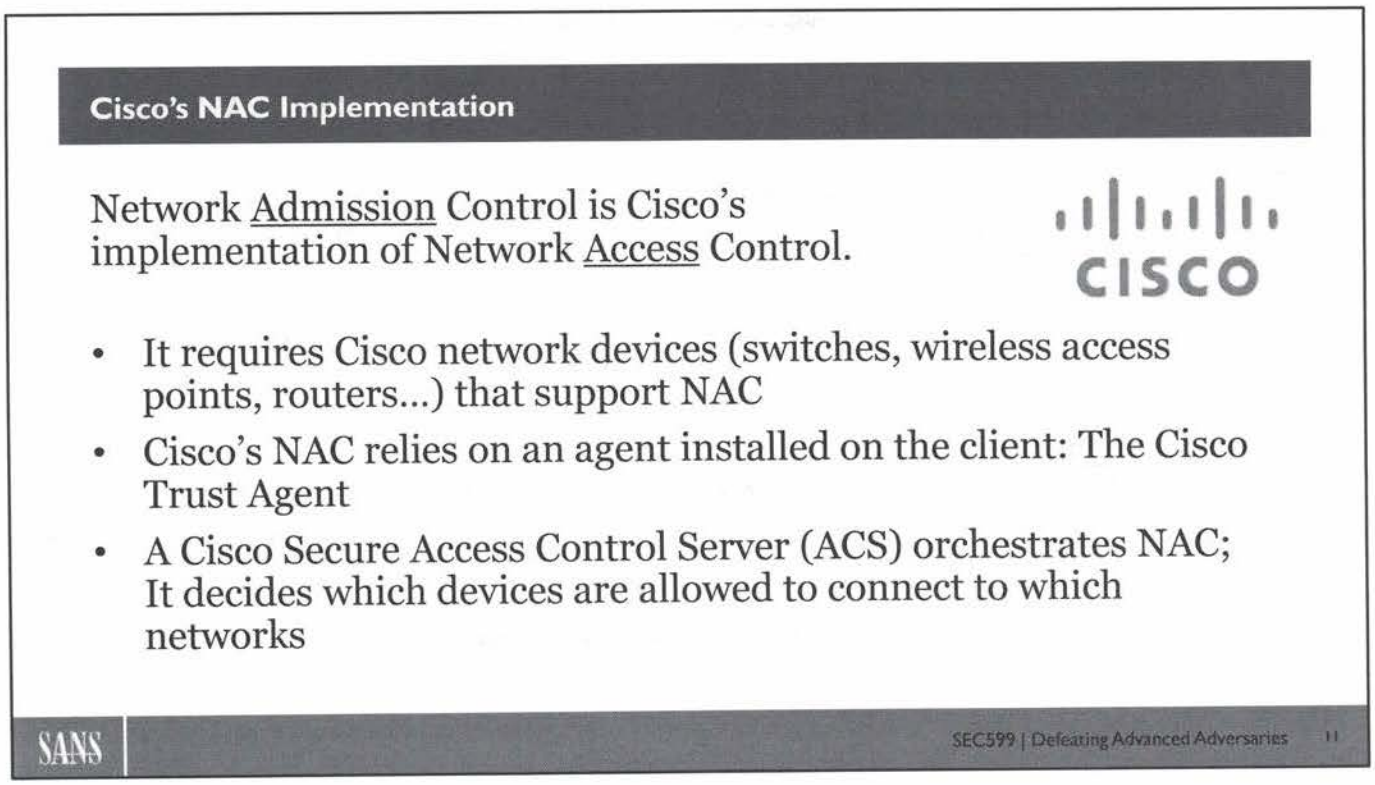
When a client connects to the Ethernet network, it communicates with a switch. Typical switches will immediately start to switch the network packets from and to the client to other network devices.

This is not the case when NAC is implemented. With NAC, special switches are used that support NAC. These switches will first identify and authenticate the client that connects to the switch (we will see later in details how this can be done). If the client is not authorized to connect to the network, the switch will not forward any traffic from the client: the client is blocked from accessing the network. It can only negotiate network access with the switch, all other traffic is not allowed.

If the client is authorized to access the network, the switch connects the client to a VLAN that gives it access to the corporate network.

As explained, there are enterprises that use NAC to enforce security policies like antivirus and patching. In these networks, a NAC agent running on the client will inspect properties of the operating system, like the presence of an up-to-date antivirus engine and the level of Windows patching.

If a device is recognized as a corporate device, but the NAC agent reports that it does not comply with corporate policies, then the switch will not give it access to the corporate network. Instead, the switch will put the noncompliant device in the VLAN of the remediation network. The remediation network is a network with very limited access. It has no access to the corporate network, but it contains servers to which the client can connect to become compliant by updating its antivirus signatures and applying missing patches. Once a machine is up-to-date, it can start the NAC process again and be connected to the corporate network.

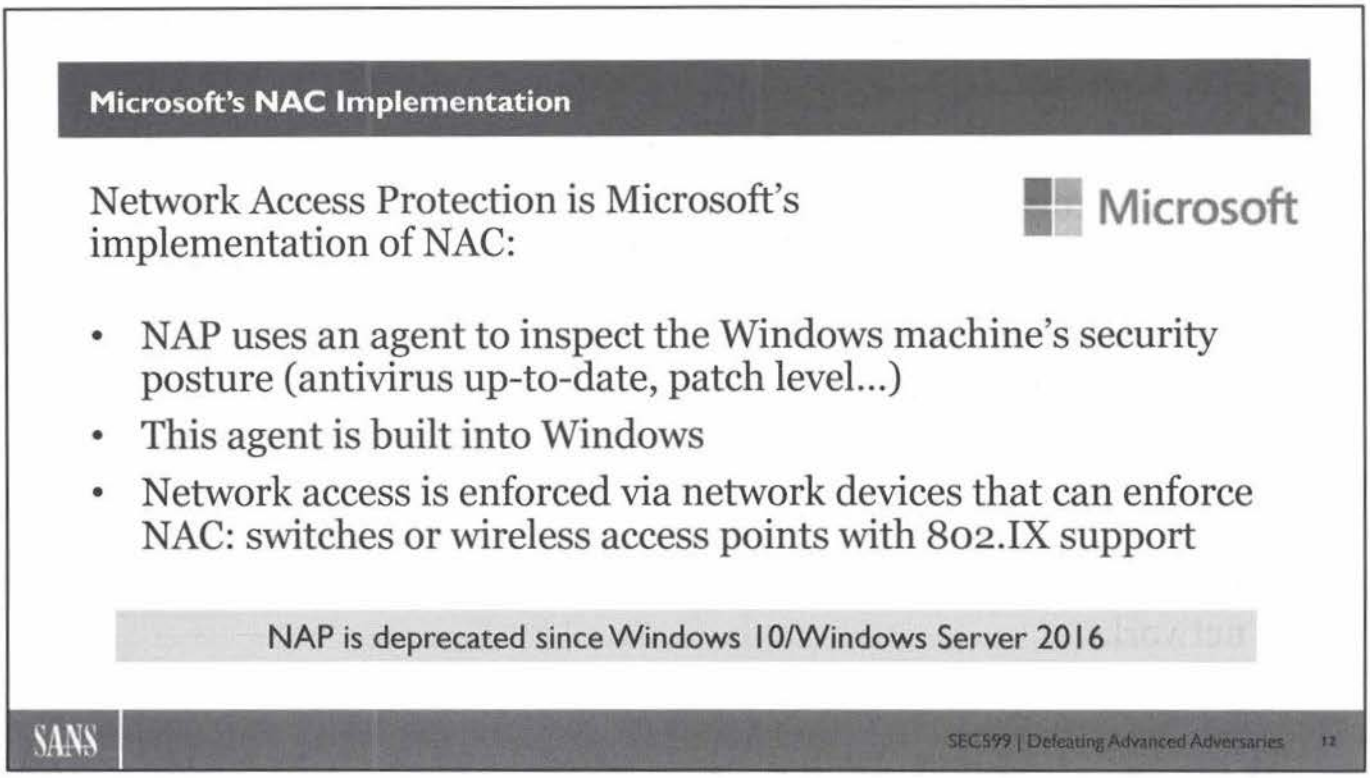



**Cisco's NAC Implementation**
The networking company Cisco provides Network Access Control in some of its products. Cisco calls this Network Admission Control.

Network Admission Control is implemented in various devices and software sold by Cisco. For Ethernet networks, Cisco switches with NAC support are required. For Wi-Fi, Cisco wireless access points with NAC support are required. They will enforce the NAC policies.

Cisco provides a client agent (the Cisco Trust Agent, software to be installed on the operating system of the machine) that will interact with Cisco's NAC aware devices and inspect the machine for various security properties (like antivirus, patches…).

This agent interacts with the devices that will provide access to the network: a switch or a wireless access point. The agent participates in the authentication and provides information on the security posture of the device. The switch or access point relay that information to a Cisco Secure Access Control Server. This ACS device decides which devices authenticated properly and are thus, eligible for network access, and which devices are not. Based on the security posture of the authenticated device, the ACS will decide if the client is given access to the corporate network, or to a remediation network. These decisions are passed on to the switch or access point, which enforce the decision.



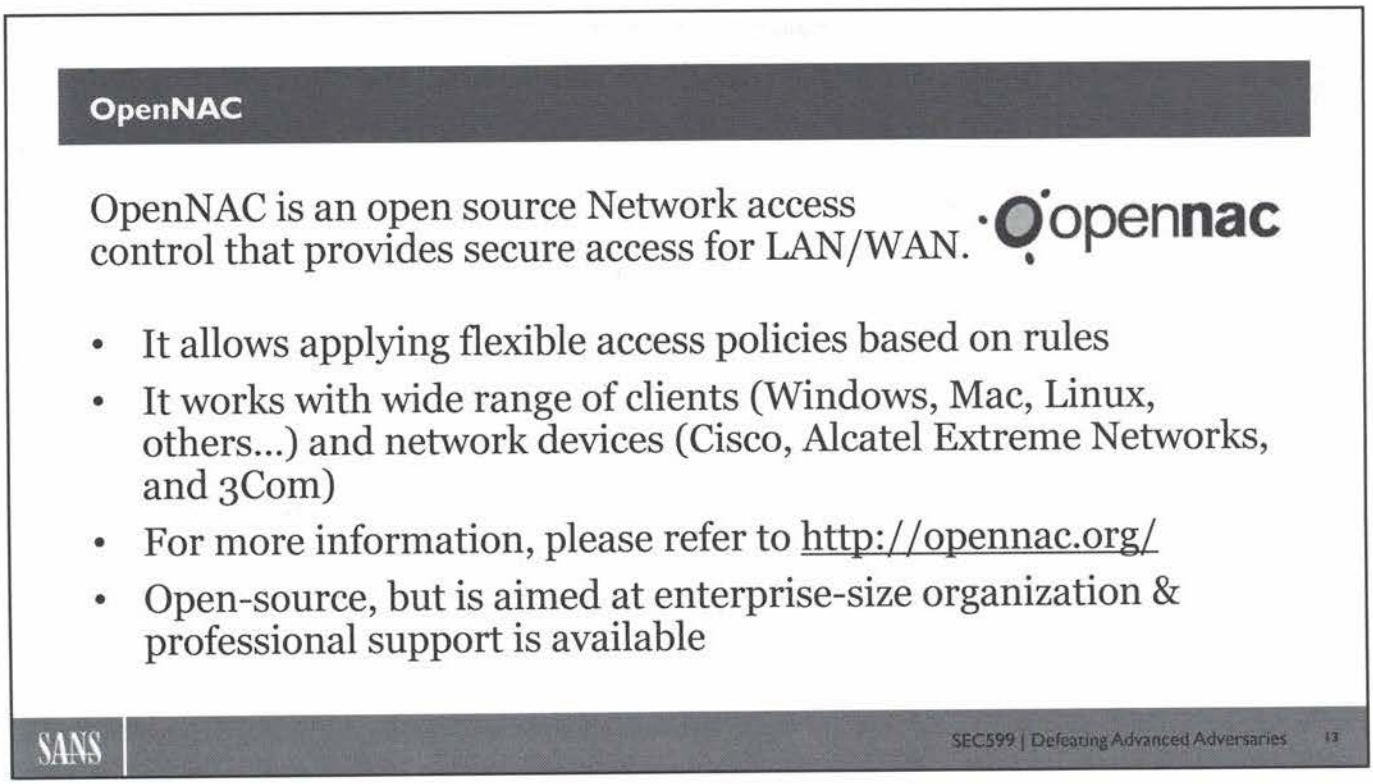

**Microsoft's NAC Implementation**
Microsoft's implementation of NAC is called Network Access Protection (NAP).

NAP too relies on an agent to inspect a Windows machine; does it have an antivirus installed and are the signatures up-to-date; are recent patches installed; is the local firewall enabled…? The agent can answer all these questions and report the information to a Windows Server with NAP support.

This agent does not require separate installation, it is an integral component of Windows.

The network access itself is enforced by network devices (not sold by Microsoft). These switches and wireless access points must support IEEE's 802.IX NAC standard (discussed in the next slides). They relay the information from the agent to the Windows server, which makes the decisions.

NAP is no longer available in the latest versions of Windows. It was deprecated in Windows Server 2012 R2 and is not available in Windows 10 and Windows Server 2016.

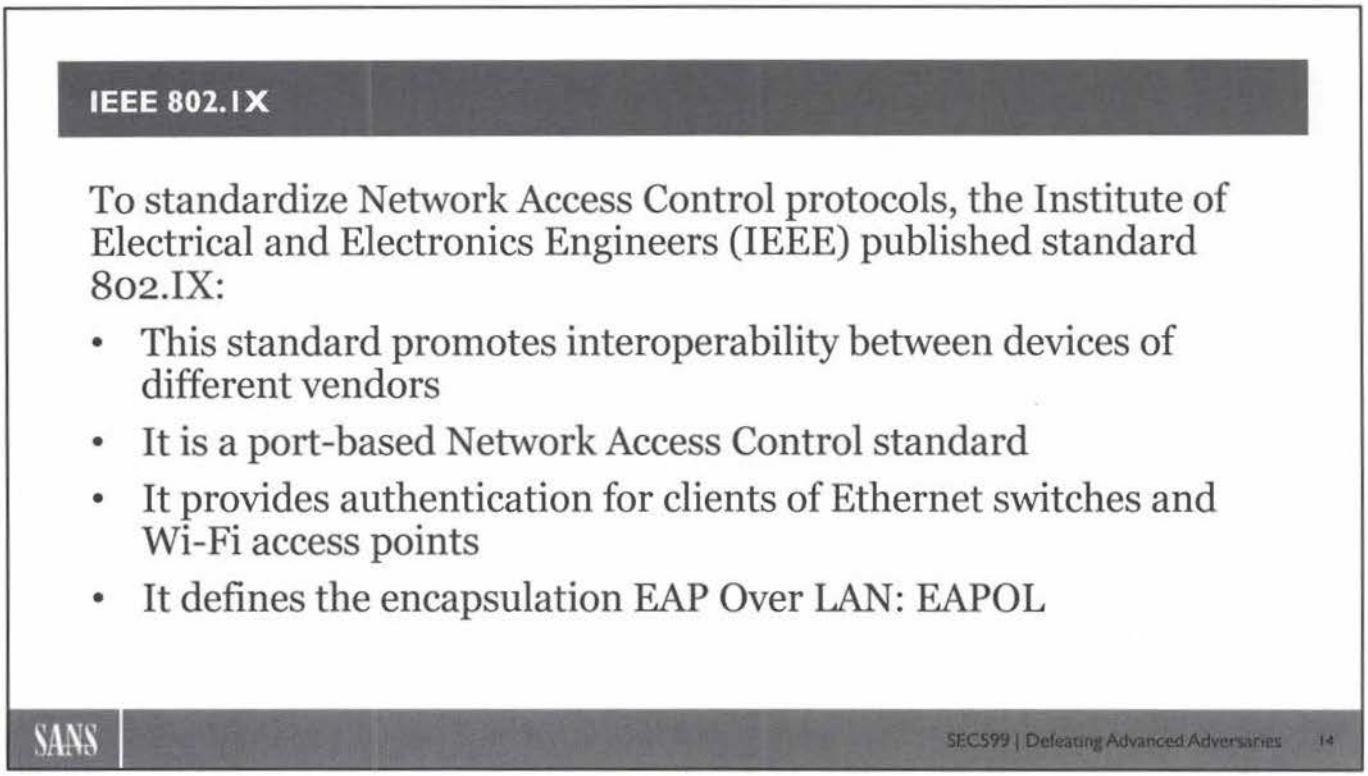



**OpenNAC**

OpenNAC is one of (very) few open source Network Access Control (NAC) solutions. It supports both LAN & WAN. Some of its key features include:

- Supports a wide variety of clients (Windows, Mac, Linux, others…) and network devices (Cisco, Alcatel & 3Com);
- Arranges corporate network access based on a set of rules (access policy);
- The availability of Notifications or Quarantine to users regardless of the client device (via browser);
- Value-added services such as monitoring, discovery, and configuration of network infrastructure;
- Authentication backend based on LDAP or AD;
- Detection of OS, antivirus, firewall and OS updates of devices connected to enforce access policies.

For additional information, please refer to http://opennac.org.




**IEEE 802.1X**

The problem with solutions from different vendors, is often interoperability.

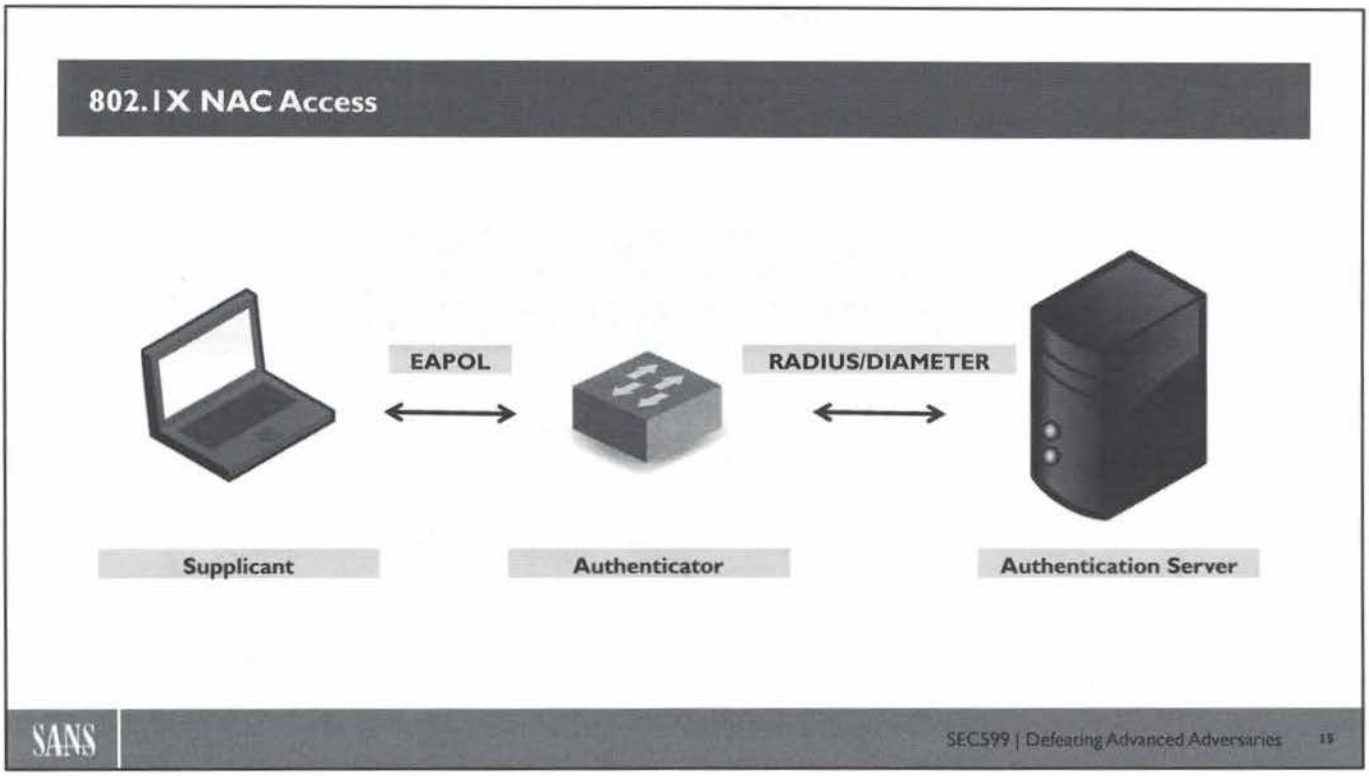
To promote interoperability of different NAC solutions, the Institute of Electrical and Electronics Engineers (IEEE) worked out standard 802.1X. This is a standard from the 802.1 workgroup which defines network protocols.

802.1X is a port-based Network Access Control standard: it uses network ports to allow or block access to the network.

This standard defines authentication protocols for both LAN and WLAN: clients can connect to switches or wireless access points and follow the same protocol.

IEEE 802.1X uses the Extensible Authentication Protocol (EAP) to authenticate clients. EAP is an authentication framework defined in RFC 3748. EAP is not an authentication mechanism, but an authentication framework: it defines a standard in which different authentication mechanisms can be used. Examples of authentication mechanisms: passwords (EAP-PWD), pre-shared key (EAP-PSK), certificates (EAP-IKEv2)…

802.1X implements the EAP protocol over LAN (802): EAPOL.



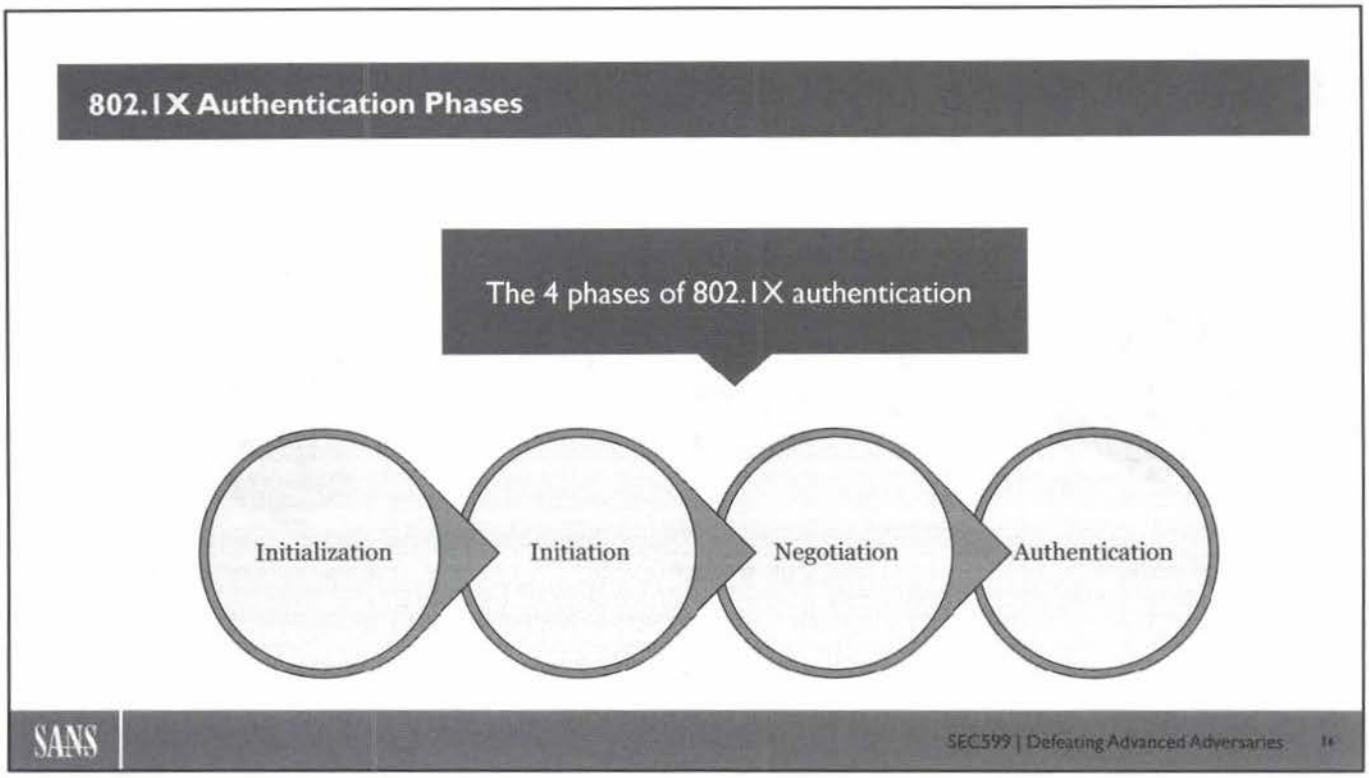

**802.1X NAC Access**
Three parties are involved in 802.1X NAC access:

- The supplicant (the client, like a laptop)
- The authenticator (the network access point, like a switch)
- The authentication server (the orchestrator, like a RADIUS server)

Authentication is done via a method supported by the Extensible Authentication Protocol. As stated before, this is not a specific mechanism but a framework that supports different authentication mechanisms. One authentication mechanism often used in corporate environments, are certificates. They require some infrastructure like a PKI, but offer several advantages: not only are they transparent to the end user, but they can be automatically deployed with Active Directory and revoked on a case-by-case basis.

The EAP protocol is encapsulated in EAPOL (EAP Over LAN) when it is used to communicate between the supplicant and the authenticator. The authenticator will relay the EAP packets to the authentication server. Depending on the implementation of the authentication server, RADIUS or DIAMETER will be used to encapsulate the EAP protocol.

The authentication server decides to allow or deny access to the supplicant, and the authenticator enforces this decision. In its initial state, the port of the authenticator will only allow EAPOL traffic. When authorized, the authenticator will allow other network traffic between the supplicant and the corporate network.




**802.1X Authentication Phases**

In a typical authentication process with 802.1X, 4 steps will be taken:

- Initialization: the port of the authenticator is in the unauthorized state, and only 802.1X traffic is allowed. All other traffic dropped
- Initiation: the authenticator periodically transmits EAP-Request Identity packets, until it receives EAP-Response Identify packets from the supplicant. The authenticator encapsulates this response into a RADIUS Access-Request packet and sends it to the authentication server.
- Negotiation: The authentication server sends a RADIUS Access-Challenge reply to the authenticator, specifying the EAP method to be used for authentication by the supplicant
- Authentication: authentication is then performed according to the selected authentication method by exchanging EAP Request and Response packets between the supplicant and authentication server over the authenticator (direct communication between supplicant and authentication server is not allowed). With successful authentication, the authentication server terminates the authentication phase with an EAP-Success message. Otherwise it terminates with an EAP Failure message. In case of an EAP-Success message, the authenticator sets the port in authorized state and normal network traffic is allowed to flow.

## Unsupported Devices

An issue with NAC is that not all devices that require a network connection, support NAC or 802.1X...
There are devices that cannot authenticate:

- Network connected printers
- IP cameras
- VOIP phones

These devices can be put in a separate VLAN that does not require 802.1X authentication, or "authenticate" based on MAC address. This separate VLAN must be properly segregated from the normal corporate LAN.



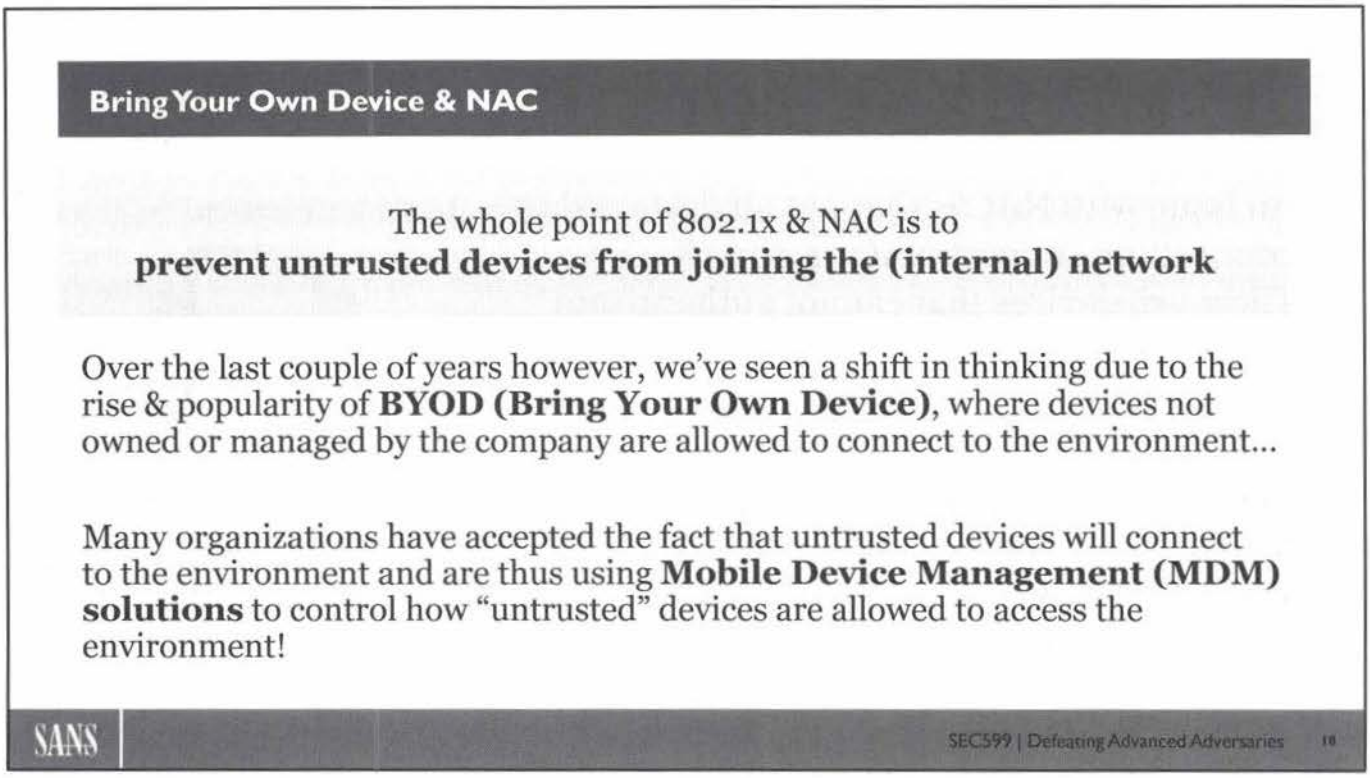
**Unsupported Devices**
NAC can help a lot with controlling access to our corporate networks, but it requires devices that understand NAC, and can authenticate for example with 802.1X.

These capabilities are often not found in low-cost devices, or in devices that have very specific functions:

- Printers and scanners that are connected to the network
- Surveillance cameras that use networking protocols to transmit images
- Voice-over-IP telephones
- …

Such devices can only be connected to ports that do not impose 802.1X authentication. As this is a clear risk opening our network for abuse, mitigations must be applied. For example, the VLAN that is used by these ports can be a dedicated VLAN with restricted access to resources, only allowing the minimum necessary for these devices to operate properly.

Another option is to use a whitelist of MAC addresses, and only allow devices whose MAC address is on the whitelist. Remark that for an advanced adversary, this is not a difficult obstacle: MAC addresses can be easily spoofed.

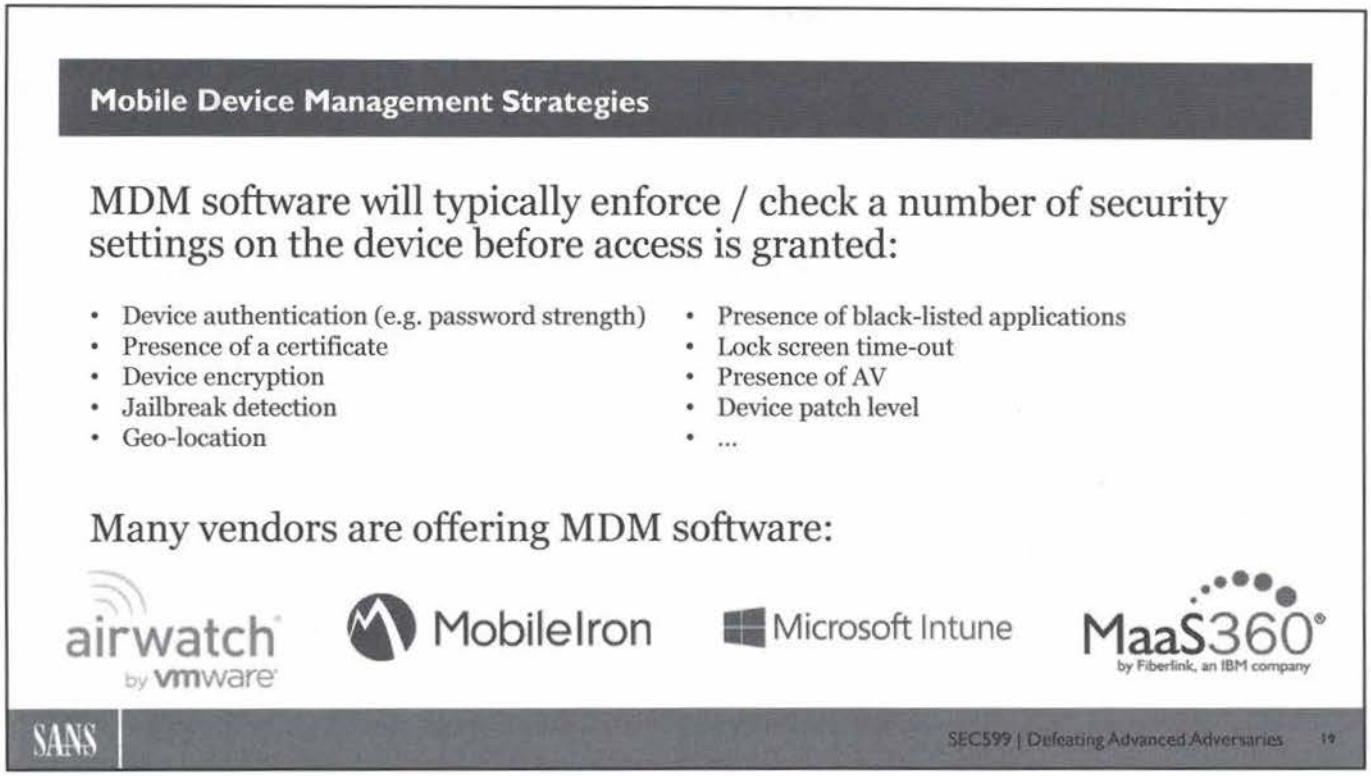



**Bring Your Own Device & NAC**

The whole point of 802.1x & NAC technologies was to prevent untrusted devices from joining the (internal) network. This was an effective strategy during the era where employees only relied on devices provided & managed by employers. In such a case, it was "easy" to enforce certain security controls & measures, as all devices were known up front and they could be fully managed by employers.

Over the last couple of years however, we've seen a shift in thinking due to the rise & popularity of BYOD (Bring Your Own Device), where devices not owned or managed by the company are allowed to connect to the environment. This is often considered for cost-saving reasons, or just to provide additional flexibility to employees. This does however affect our security controls: we can no longer assume that all devices which will connect to our environment are owned & controlled by us.

Many organizations have accepted this fact and are relying on Mobile Device Management (MDM) solutions to control how "untrusted" devices are allowed to access the environment!




**Mobile Device Management Strategies**
Mobile Device Management (MDM) software will typically enforce / check a number of security settings on the device before access is granted:

- Device authentication (e.g. password strength, PIN code, lock-out…)
- Presence of a certificate
- Device encryption
- Jailbreak detection
- Geo-location
- Presence of black-listed applications
- Lock screen time-out
- Presence of AV
- Device patch level

These settings can be used in different ways:

- The MDM solution could enroll devices and then enforce the settings (will require acceptance of the owner of the device) prior to allowing access to the environment;
- The MDM solution could simply validate the current settings on the device (without enforcement) and decide whether or not to allow access to the environment;

Specifics depend on the actual MDM solution being used, it's worth noting that many vendors currently offer MDM solutions (Airwatch, MobileIreon, Intune, MaaS360…)

## Network Access Control, 802.1X & MDM Summary

IEEE 802.1X is a port-based network access control standard which can help us protect our network from unwanted devices, different vendors offer different solutions, use the one that is tailored to your environment.

NAC / MDM software can also be used to improve the security posture of corporate devices (enforce security policy settings).

Not all devices support NAC or MDM software, so exceptions will exist and need to be managed.



**Network Access Control, 802.1X & MDM Summary**

802.1X is an IEEE standard for port-based NAC (PNAC). NAC can be a great help to secure corporate networks. But it comes with a non-negligible cost: NAC infrastructure requires network devices and clients that are compliant with NAC standards. Authentication failures and devices that are not NAC-aware must be managed. Several vendors, like VMware, Cisco, Microsoft, IBM… offer NAC solutions. These have evolved over time and are more and more being advertised as MDM solutions.

An extra benefit of NAC, is that it can be used to increase the security posture of devices that connect to the network. This is becoming more and more common practice with the rise of BYOD (Bring Your Own Device) and MDM software. Devices that do not meet certain security policy settings will be isolated until they comply with the enforced policy settings.

Unfortunately, not all devices support NAC (e.g. printers, Internet of Things…) and extra measures must be taken to accommodate these devices in a secure way.

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

**Securing the Network**

Network Access Control & 802.1X

**Securing Software**

Software Development Lifecycle (SDL) & Threat Modeling

Vulnerability Assessments

Exercise: Authenticated Scans Using Nessus

Patch Management

Exploit Mitigation Techniques

Exercise: Exploit Mitigation Using Compile-Time Controls

Exploit Mitigation Techniques – Exploit Guard, EMET, & Others

Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**

OS Hardening & Best Practices

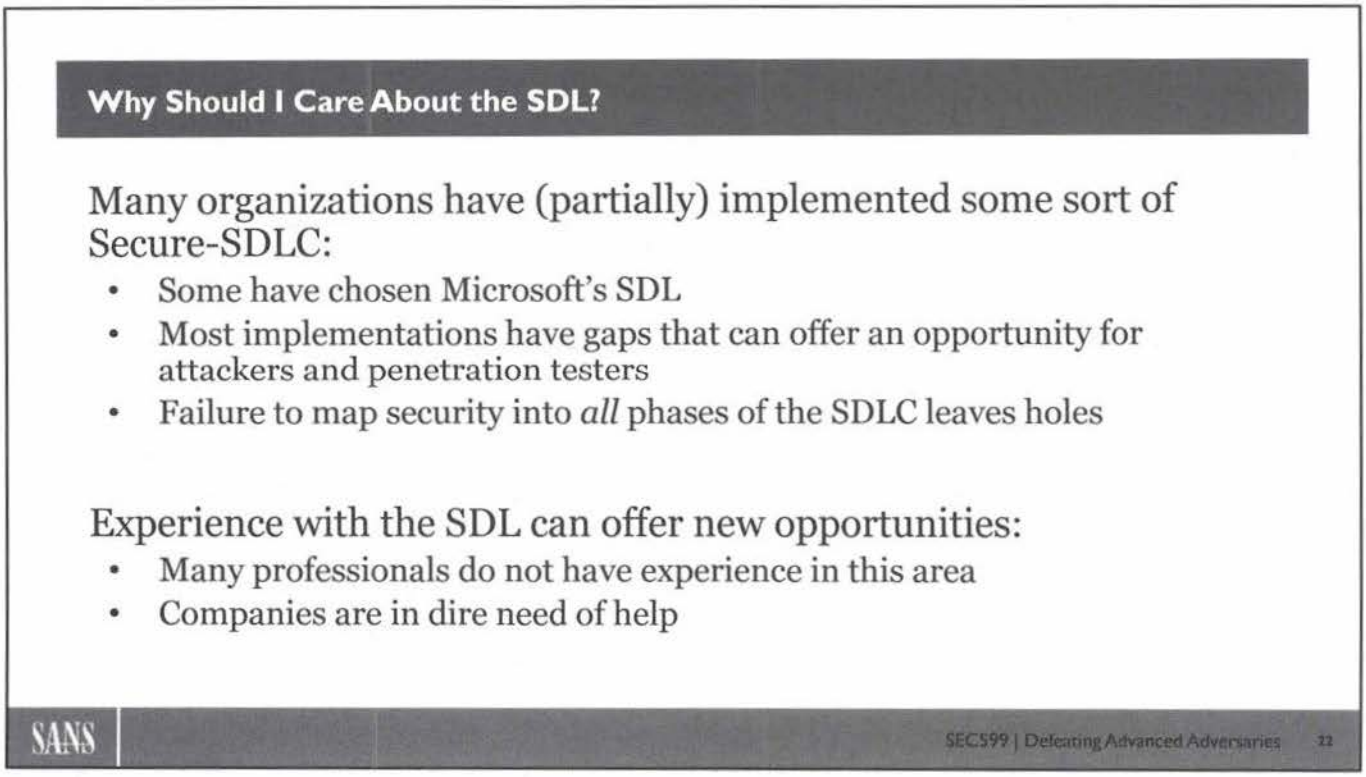
Endpoint Protection Solutions

Application Whitelisting to Stop Payload Execution

Exercise: Configuring AppLocker



This page intentionally left blank.




**Why Should I Care about the SDL?**

The question occasionally comes up about why non-developers concern themselves with Microsoft's Security Development Lifecycle (SDL) or a Secure-SDLC. The better you understand how organizations write their code, the easier it is to identify potential areas of weakness. If an organization does a good job performing peer review and static analysis, but lacks dynamic testing during the validation phase, it should be called out as a gap. This gap allows us to prioritize our time on the areas with the biggest potential for concern. Most organizations have implemented some sort of security into their development process; however, many are severely lacking. Failure to map security into each and every phase of the SDLC leaves holes, which can be exploited.

The SDL is still a relatively new concept for most companies and many are in need of help. Experience in this area can offer new job opportunities to a professional with the proper skills.

## Microsoft Security Development Lifecycle (SDL)

Initiative started sometime in 2002 – 2003:

- Based on a memo in January, 2002 from Bill Gates known as the Trustworthy Computing (TwC) memo
- Applications to be built with security from the ground up

First version of the MS SDL made public in 2008, Version 3.2:
http://www.microsoft.com/en-us/download/details.aspx?id=24308

Version 5.2 available as of May, 2012:
http://www.microsoft.com/en-us/download/details.aspx?id=29884

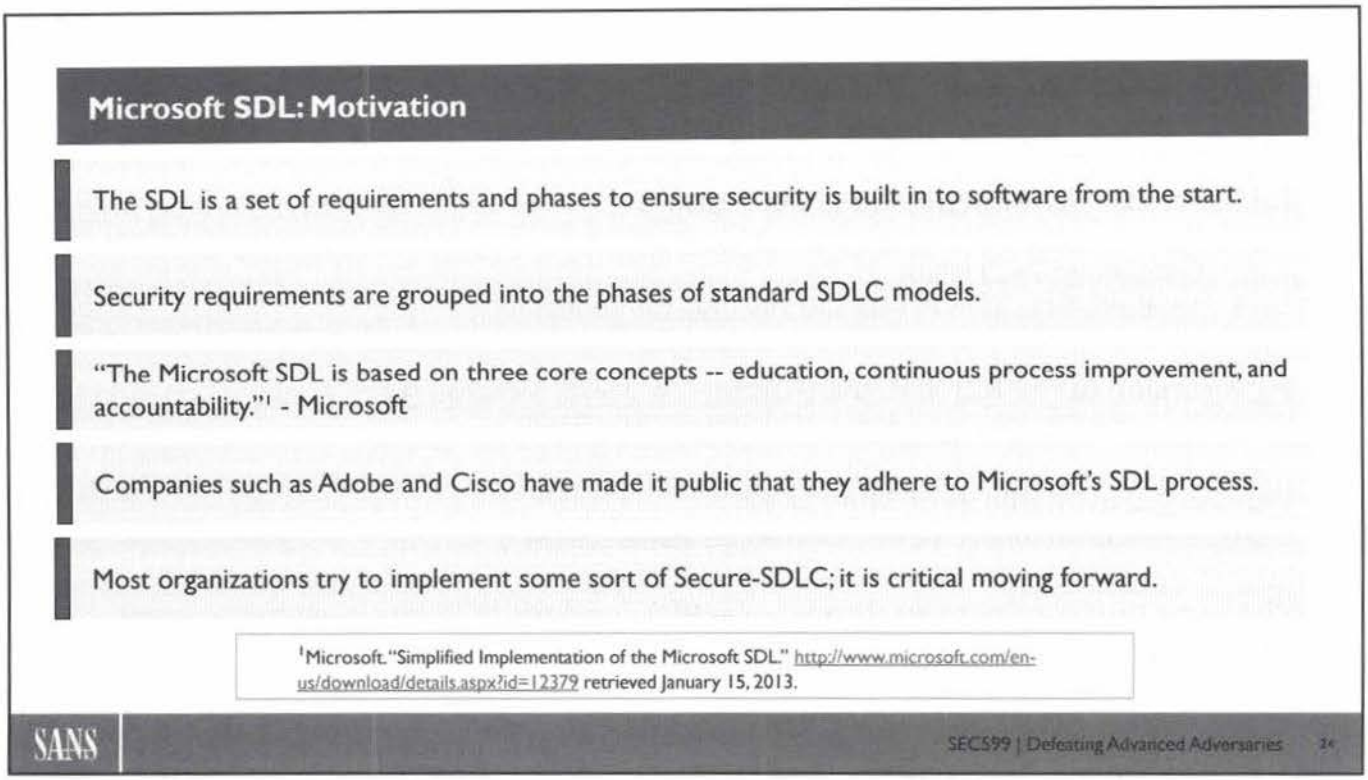
Vista was the first OS to go through the SDL, and the SDL has been mandatory since 2004.



**Microsoft Security Development Lifecycle (SDL)**
The Microsoft Security Development Lifecycle (SDL) was started sometime in 2002 – 2003 to ensure that applications and operating systems are built with security from the ground up. Microsoft's SDL information webpage can be found at http://www.microsoft.com/security/sdl/default.aspx. This was during a time when Microsoft was dealing with major security issues from various pieces of malware, such as the Melissa Virus, as well as high-profile legal battles around web browser monopoly with Internet Explorer packaging. On January 15, 2002, Bill Gates sent out a memo known as the Trustworthy Computing (TwC) memo. The memo described major changes that needed to occur to ensure Microsoft and its customers were protected and that they could rely on the operating systems. The memo from Bill Gates can be read at http://www.wired.com/techbiz/media/news/2002/01/49826.

The first known version of the SDL (Version 3.2) was released to the public in 2008 and can be downloaded at http://www.microsoft.com/en-us/download/details.aspx?id=24308. Version 5.2 was the latest available version at the time of this writing and is available at http://www.microsoft.com/en-us/download/details.aspx?id=29884. Some great introductory material and presentations on the SDL are available at http://www.microsoft.com/en-us/download/details.aspx?id=16420. Microsoft Vista was the first full operating system to go through the SDL process. Microsoft also used the process to retroactively go through prior versions of code.

The contents of this module are heavily based on the Microsoft Security Development Lifecycle (SDL) and STRIDE threat modeling processes, as well as the author's experience with the implementation of Secure-Software Development Life Cycle (S-SDLC) programs in various organizations. The material written for this module references and leverages the concepts and ideas behind these models. More information on these processes can be found at http://www.microsoft.com/security/sdl/default.aspx and http://msdn.microsoft.com/en-us/library/ee823878%28v=cs.20%29.aspx.

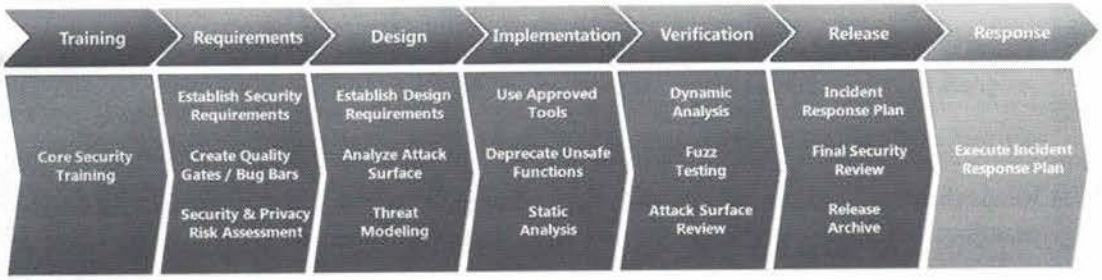



**Microsoft SDL: Motivation**

The Microsoft SDL is a detailed security process that must be adhered to during software development. It provides a specific group of activities to be performed during each phase of a Software Development Life Cycle (SDLC) to ensure that security is built into software from the beginning. Microsoft has various core concepts, some specific to Microsoft's deployment, such as ensuring the use of automated tools for finding bugs and other issues, tools for compliance tracking, and tools to help program managers evangelize the use of the SDL throughout various divisions and teams within the organization. Steve Lipner, Director of Security Compliance at Microsoft, has done quite a few presentations explaining how the SDL is deployed in Microsoft. Some of it can be applied to many organizations, whereas other practices are specific to Microsoft. Each organization must cater the process to their development program. During Lipner's presentation at OWASP's AppSec conference in 2010, he claimed that updates to the SDL are made only once per year and are mainly focused on the creation of new tools used for automation and fuzzing.

Companies such as Adobe and Cisco Systems have publicly stated that they adopted some or all of the Microsoft SDL. Regardless of whose process your organization adopts, the use of an overall secure SDLC is essential in this day and age.

## Phases of the MS SDL



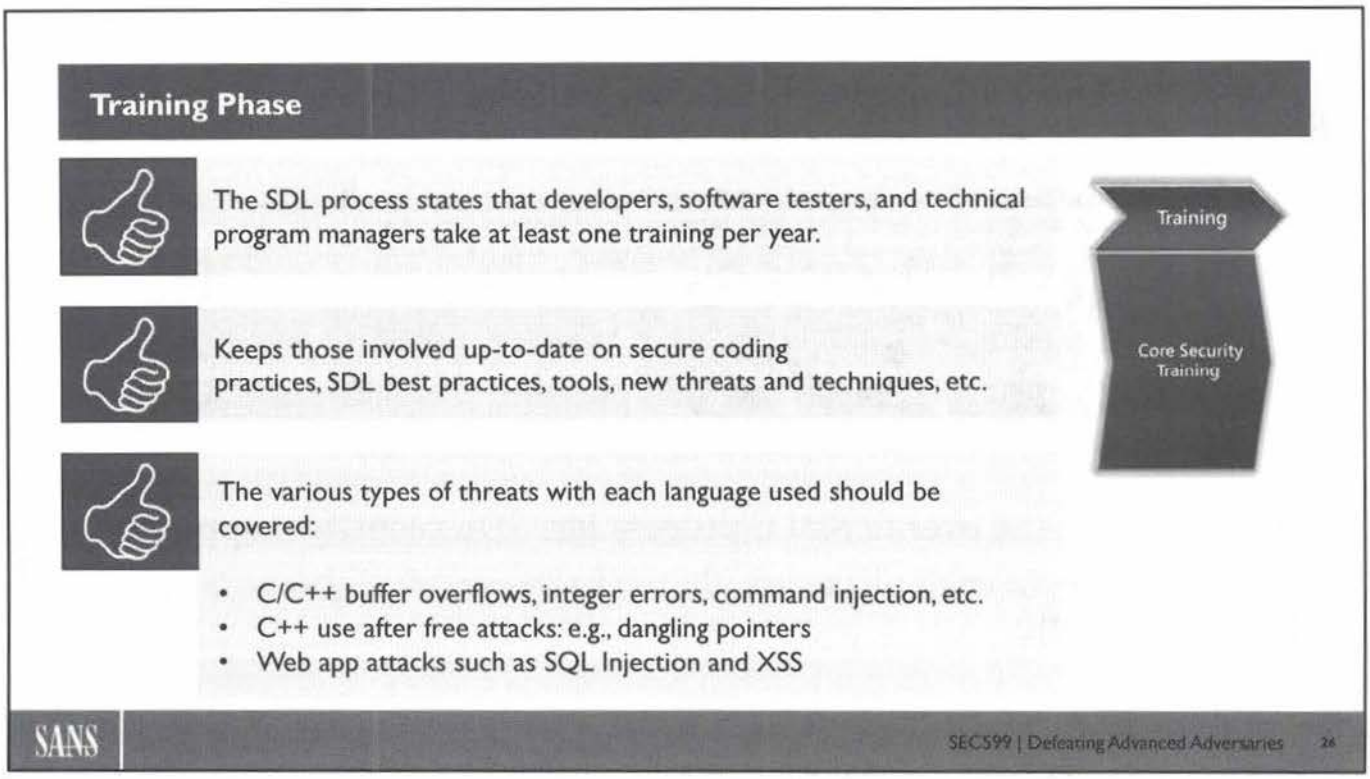

http://www.microsoft.com/security/sdl/discover/default.aspx

Each phase of the overall SDLC process has SDL security mappings. The mappings to each phase are shown in the graphic above, taken from Microsoft.



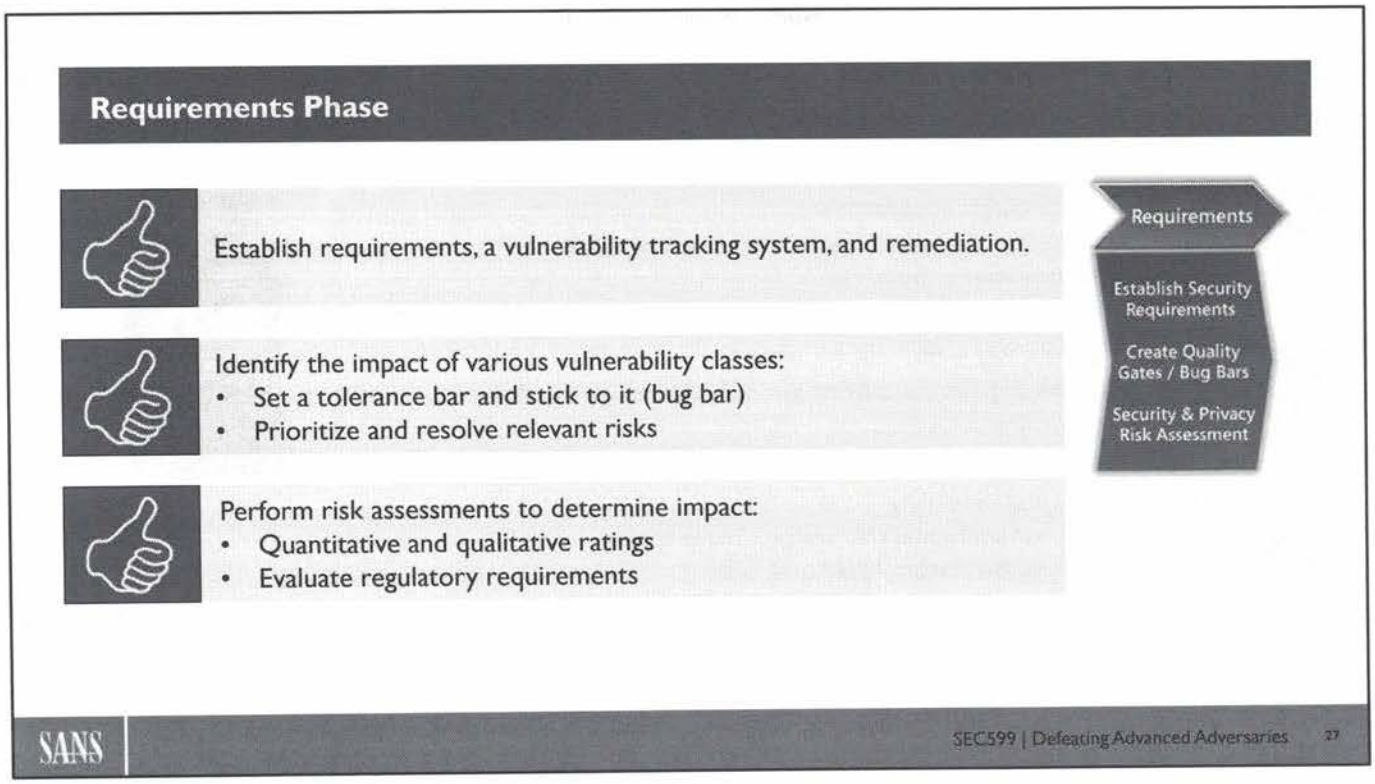
**Phases of the MS SDL**

Each phase of the high-level SDLC processes listed on this slide has associated SDL security mappings. The diagram in the slide, taken from Microsoft, shows the mappings that are covered in the following slides.




**Training Phase**

According to Microsoft, leveraging the previously documented references, the SDL states that all software developers, testers, and relevant technical program managers are to attend at least one security training per year, covering each phase of the overall SDL process and specific threat types and techniques used for modeling. This requirement helps ensure that each member is up-to-date on his organization's implementation of the SDL process, new tools, threats, and commonly used techniques. The training should include any relevant languages used by the developers and it addresses vulnerability classes associated with those languages. Developers in C and C++ get training on buffer overflows, function pointer overwrites, integer errors, format string attacks, information leakage, use after free attacks, and many others. Web application developers receive training that is more focused on attacks, such as SQL injection, cross-site request forgery (CSRF), cross-site scripting (XSS), and others. Threat modeling and risk assessment techniques are also included in the training programs.

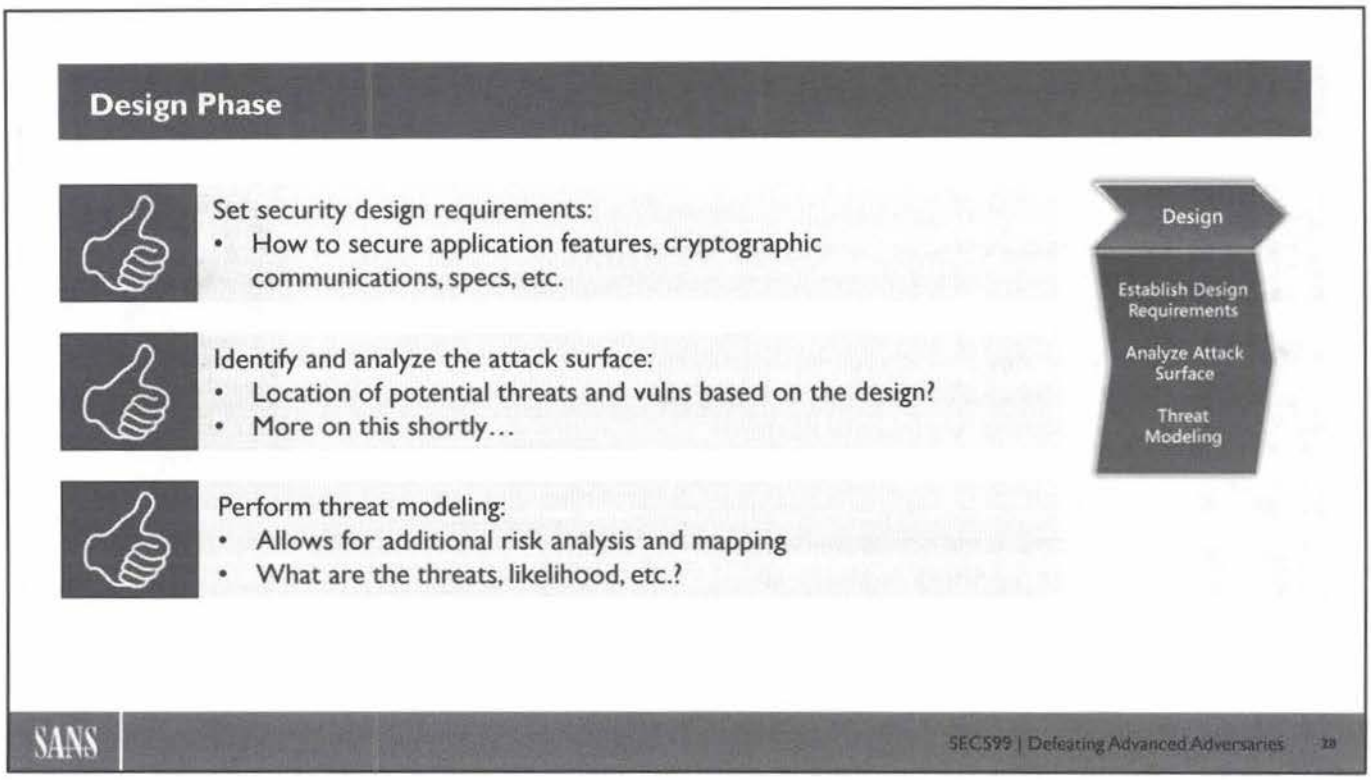



**Requirements Phase**

During the requirements phase, there are three main areas to cover. The first area is to establish the security requirements and to build a security team and the processes assigned to the project. This includes assigning security staff, creating a tracking system for bugs, creating a remediation process, and establishing the criteria for any privacy requirements around the data elements involved. Introduction of these items early on will help ensure a smooth process flow.

The next area is to set a tolerance bar or threshold that must be adhered to in order for the software to go into production. This typically falls in line with an organizational risk assessment process. During most risk assessment processes, the primary goal is to document all risk items, map them to policy and regulatory violations, set the initial risk rating, map in any potential mitigations, and document the likelihood and the potential impact to the organization. The difference is that with the SDL, you are setting a threshold that cannot be exceeded. This means that if it is determined during the requirements phase that no medium or high risks are permitted to go into production, they must be fixed prior to release.

The third area is centered around the identification of software features and functionality and mapping those functions to areas of concern, such as with consumer privacy, data protection, and regulatory requirements. It basically serves as a risk assessment on specific areas of the application.

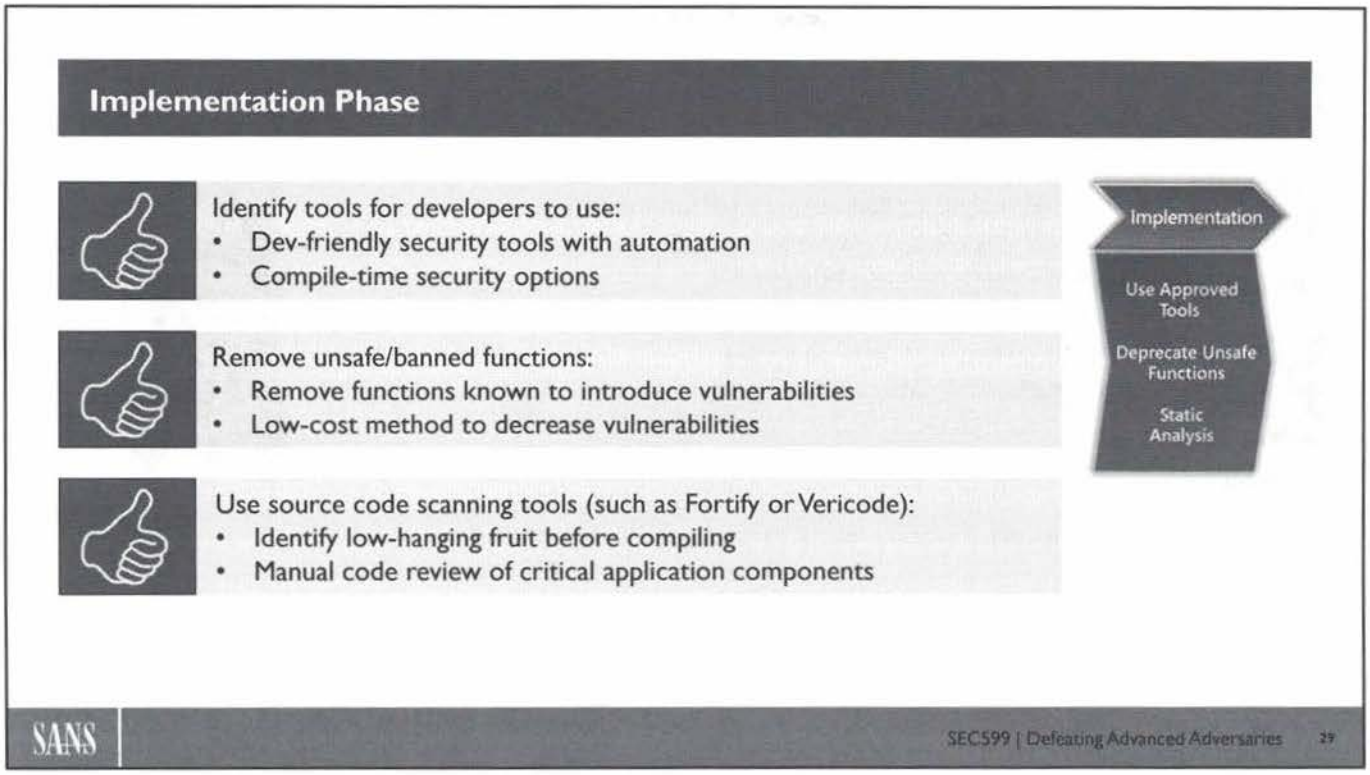



**Design Phase**

During the design phase, three main areas are covered. First, design requirements must be established. Leveraging the requirements phase, software features and functionality must be written to adhere to all privacy and security requirements. Because we are looking specifically at privacy requirements, secure communication and storage are major considerations. Cryptographic design must be well thought out for secure implementation and the types of cryptographic attacks must be well understood. This is a good spot to add in some peer review.

Next, we want to thoroughly understand and analyze the attack surface. We must look at the design from a high level all the way down to a low-level and identify all of the potential areas where vulnerabilities may exist and map threats to those vulnerabilities. We perform this task during the design phase so that we can make changes prior to any development as a cost-saving measure, and hopefully a time-saving measure. We will get into more on attack surfaces shortly.

Finally, we flow from analyzing the attack surface into threat modeling. This gets specifically into mapping the actual attacks to the attack surface. It is okay to get creative with the types of attacks during this phase as the team should be encouraged to think outside of the box. Threat modeling is also covered shortly.

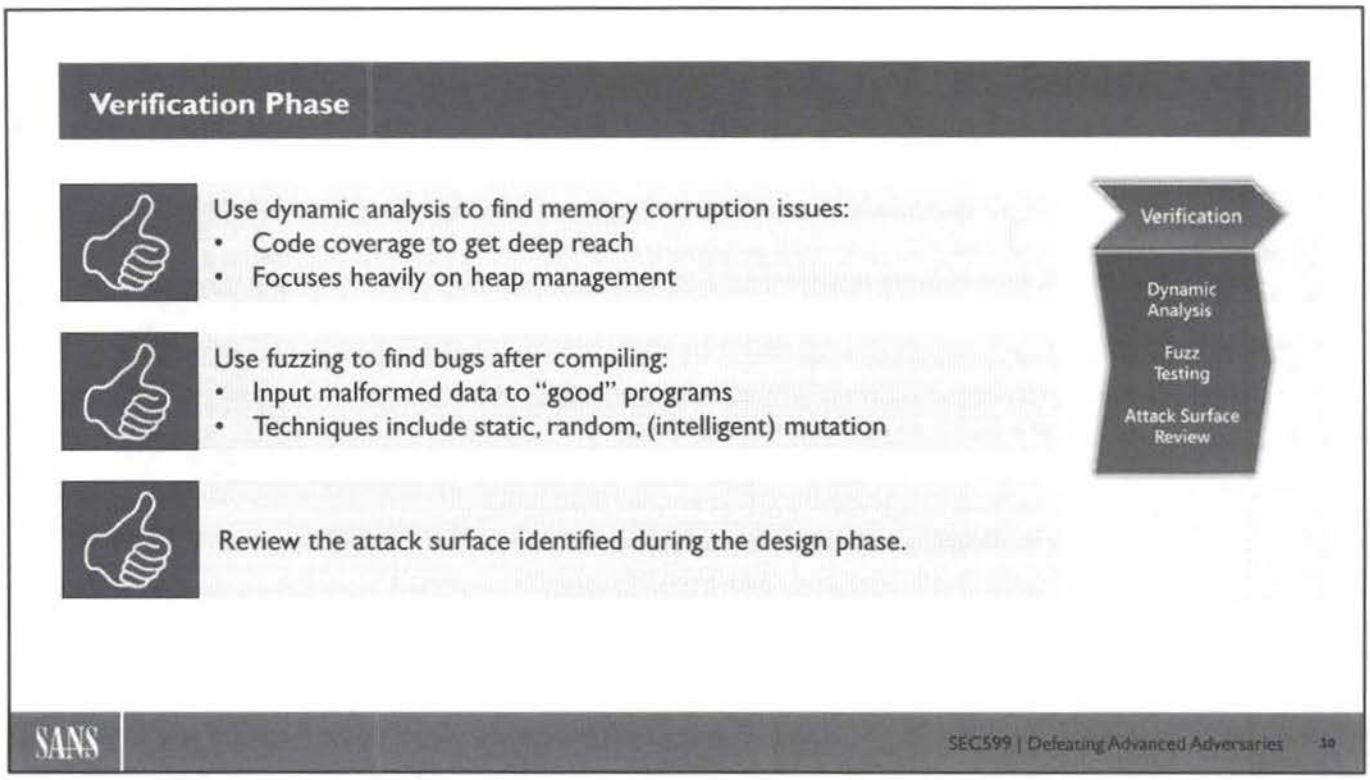



**Implementation Phase**

The implementation phase also has three main areas. The first area is centered around the use of approved tools. Security researchers and engineers along with developers should work together on solutions to help automate as much of the SDL as possible, without sacrificing security. Not every developer can be expected to be security experts and to get better support for the SDL, automation is highly desirable. Penetration testers, security engineers, incident handlers, and developers are all good resources to help identify the types of tools and exploit mitigation protections that should be used by the compiler. Depending on the target operating system where the software will be installed, compiler options such as support for address space layout randomization (ASLR), SafeSEH, stack and heap canaries, data execution prevention (DEP), and others should be designated as requirements. These types of security options and exploit mitigation controls are constantly changing and should be followed closely.

The next area is to identify any unsafe functions and add them to a list of banned functions that can be easily referenced and enforced. If possible, automating the discovery of banned functions during code review should be implemented. It is also common for operating system developers to remove unsafe functions that could cause issues if not identified during the development process. Microsoft removed support for certain functions that allow for the modification of DEP settings on Windows 7. It is important to track these types of changes.

Finally, a code review should be performed prior to compilation. Prior to the creation of automated source code scanning tools, manual review is required. This is a time-consuming process, and the chance of the reviewer missing vulnerabilities is quite high. Not all languages are supported for review; however, the bulk of the primary languages are supported by various tools such as Fortify and Veracode. In this author's experience, many of these tools are good at catching the low-hanging fruit, but they can have a bit more difficulty identifying more complex vulnerabilities. There are often a large number of false positives that must be removed prior to reaching any real areas of concern. When scanning large source code files, there can sometimes be thousands of possible vulnerabilities identified, with the majority not being a real concern. This can be frustrating for developers who are trying to ensure their code is secure and it is often better to have a separate code review team who can help remove some of the burden.

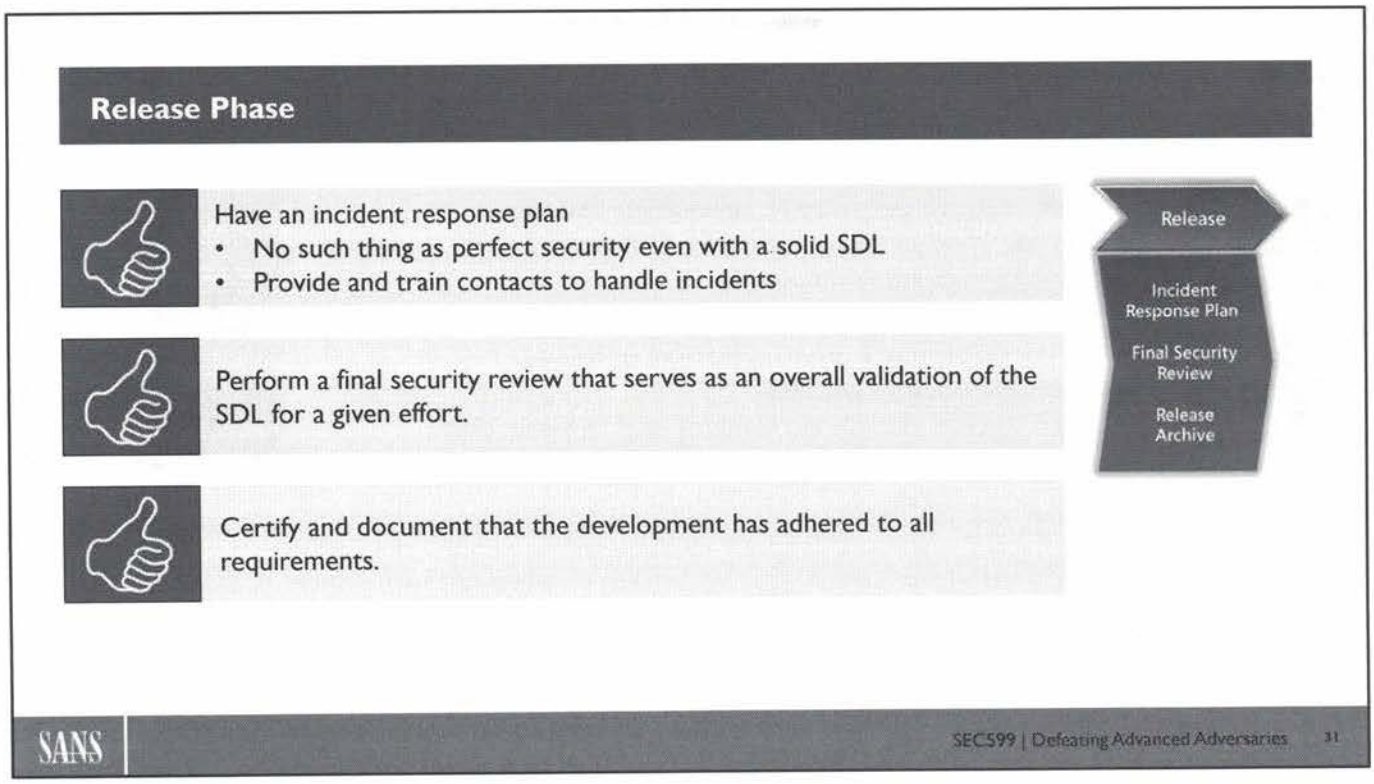



**Verification Phase**

There are three main areas of focus during the verification phase. This phase occurs after the source code has been compiled into an object file. At this point, regardless of how well you think you know your code, it has changed. The type and version of the compiler, the compiler and linker options used, exploit mitigation controls used, and other options used can heavily influence how your code will look on the other side. It is with this understanding that we must find ways to test all areas of input to the application and get good code coverage. Dynamic analysis tools and fuzzing tools have a similar role and the terms are often used synonymously. Dynamic analysis focuses more specifically on identifying runtime errors, dynamic memory corruption, user privileges and rights, and some other specific areas. This often includes using dynamic tools to input data into the application and monitor behavior.

Fuzz testing or fuzzing is a useful software testing technique that involves sending malformed data to protocol implementations based on RFCs and documented standards. Programs are built to function, preferably based on standards that allow for interoperability with other vendors' products. As we know, programs can be built in many different languages using a combination of many different functions. If you have 100 developers write the same application, each one will likely be different at the source level. Fuzzing requires you to think of all of the ways that a developer could have written a piece of software and test for relative vulnerabilities. It is not that simple of a process, though, as many vulnerabilities are complex and difficult to predict. Take the vulnerability class called "use-after-free." This typically involves dynamically allocated objects that are freed and later referenced by code. An active pointer that is pointing to a freed object is a potential recipe for disaster. These types of vulnerabilities can be difficult to spot, especially during incremental code changes. Fuzzing can greatly increase the chances of finding such bugs. There are various types of fuzzing techniques covered in other courses, including static, randomized, mutation, and intelligent mutation fuzzing with tools such as the sulley fuzzing framework by Pedram Amini and Aaron Portnoy.

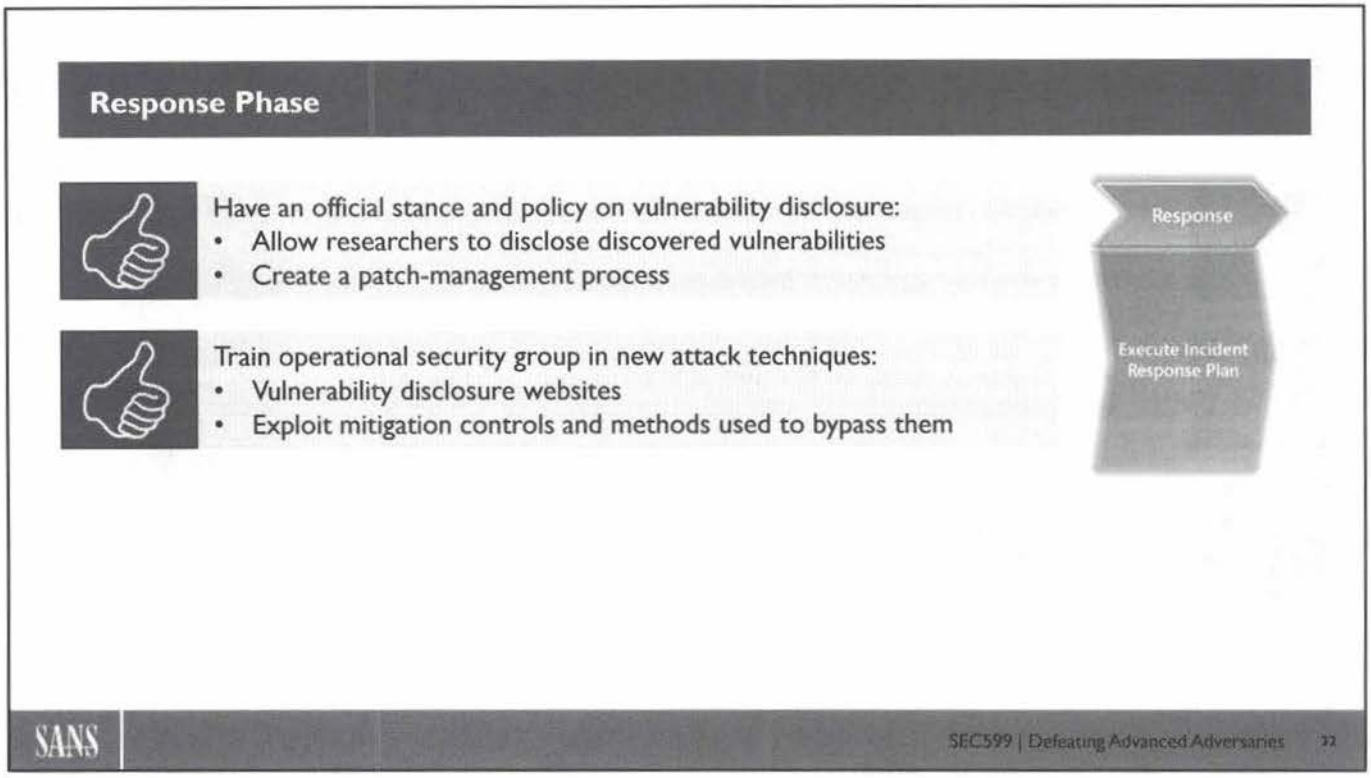
At this point, we also want to review the attack surface that we analyzed during the design phase to ensure that nothing was missed. It is fairly common for changes to occur during the actual development of the software. This allows for an opportunity to ensure all threats and vulnerabilities are captured.




**Release Phase**

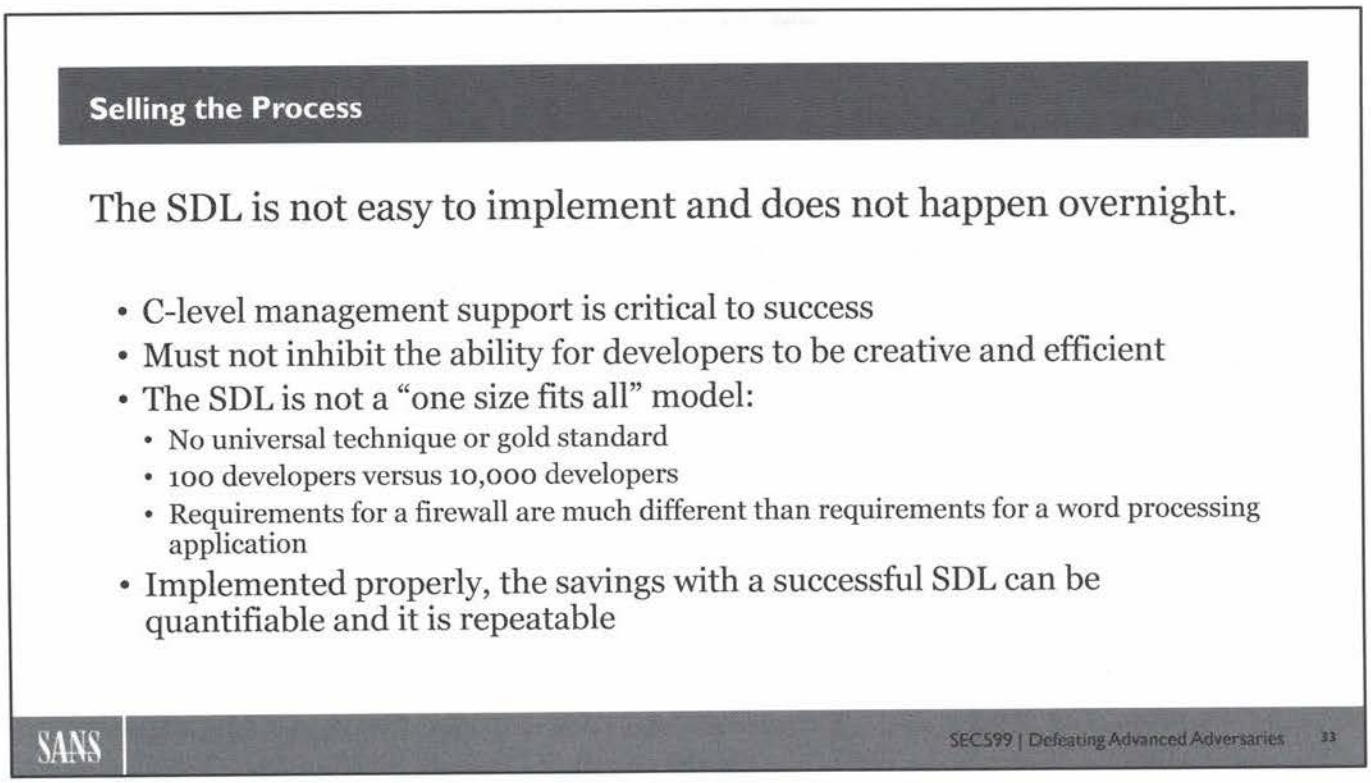
There are also three main phases during the release phase. The first is to ensure there is an incident response plan relative to the developed software. No matter how hard we try, there will never be such a thing as perfect security. If a bug is discovered, especially a critical bug, who are the main points of contact to quickly initiate a response? The answer to this question is exactly what this step is about. As per Microsoft, this step is also used to set up points of contact for inherited code. If code used was developed outside the group and questions arise, contacts should be available to answer questions, especially in lieu of training and documentation.

The second and third security review steps are in place to designate a time to take a holistic view of the SDL process to date. It works directly with the release archive step. The goal is to ensure that all phases and steps have been covered and documented. This serves as a crucial role in audits and adherence to regulatory requirements. It is this phase in which a good checklist and tracking system come in handy. The attack surface should be validated one final time, as well as threat modeling, risk tolerance as documented during the requirements phase, risk assessment, and all other steps.




**Response Phase**

The final phase is centered around the implementation of an incident response process. As stated before, there is no such thing as perfect security. No matter how mature and effective your SDL process, there will always be bugs discovered and other security issues to handle. Every organization should have a vulnerability disclosure process. There are various philosophies on how disclosure should be handled, such as full disclosure, responsible disclosure, and limited disclosure (which falls somewhere in between the other two). There should be a clear-cut process for researchers and others who find a potential bug or vulnerability in your products; even if that process says that anyone reporting bugs may face legal action. This is likely not the preferred approach, but it informs those wanting to disclose a concern about your organization's stance on disclosure. Once someone submits a finding, this is when your incident response plan goes into action. Who will respond to the individual or organization disclosing the finding? Who will take action and reach out to developers or others who should be involved? How will the submission be tracked and how long will it take to officially respond or patch the finding? How will the patch be distributed to customers if applicable? These are all processes that should be well documented and actionable.




**Selling the Process**

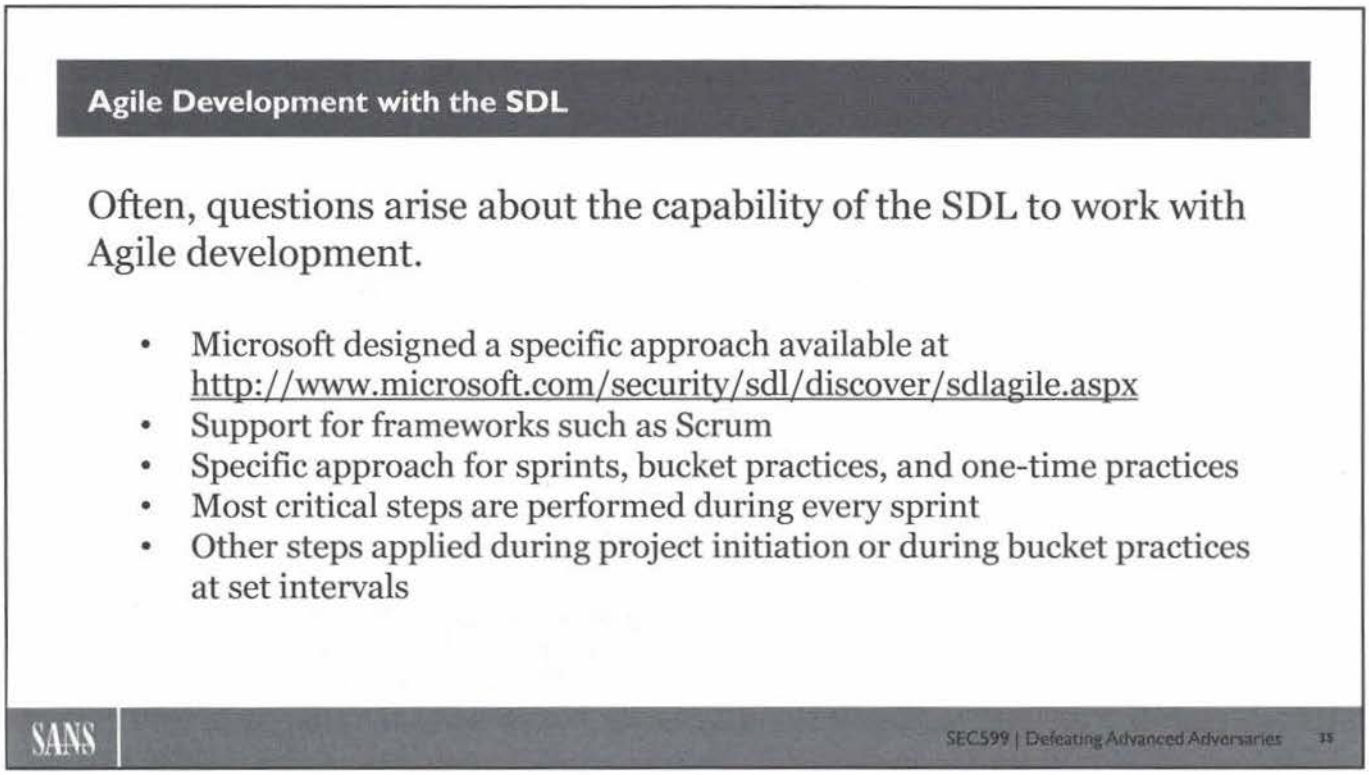
A common question is, "How can I sell this whole SDL thing to management and get support?" This is likely as hard of a task as the actual implementation of the process. In this day and age, we are all inundated with processes and the introduction of more processes can face resistance from many angles. Always remember that the ability to factor in monetary savings to the equation will almost always get some level of attention. A properly implemented SDL should do exactly that—save money. As with any other proposal, pitching the introduction of the SDL to your development process should be well thought-out and well-presented. Interviewing various lines of business from their perspective is highly beneficial. If the security operations group is burdened with incidents stemming from poor code, you want to know that information. If management is dealing with new regulations and an audit, this can also be useful information. How can you make the company's job easier and cut costs? This should be a key element when going in to pitch the process for approval.

Executive-level support for the process is critical to its success. Lacking this support will most likely result in a poorly implemented SDL or even complete failure and resistance. This should be vocalized during the proposal. One key concern that this author has learned from developers is that the SDL must not inhibit the developers from being creative and innovative. It must also not burden them down with too much process. Development can be a stressful profession with stringent requirements and sensitivity to time. Education and the ability to automate as much of the process as possible will garner more support from developers and program managers.

It must also be remembered that the SDL is not a "one-size-fits-all" model. Each organization can adopt the overall framework but must customize it to their needs. It is also not a process that can be implemented overnight, or even in a month. It takes experience and ongoing customization. A company that has 100 developers will need a different SDL application than a company with 10,000+ developers. Also, it cannot be a blanket application to all instances. A division working on the design of new firewall technology may need a different SDL than that of a word processing application.

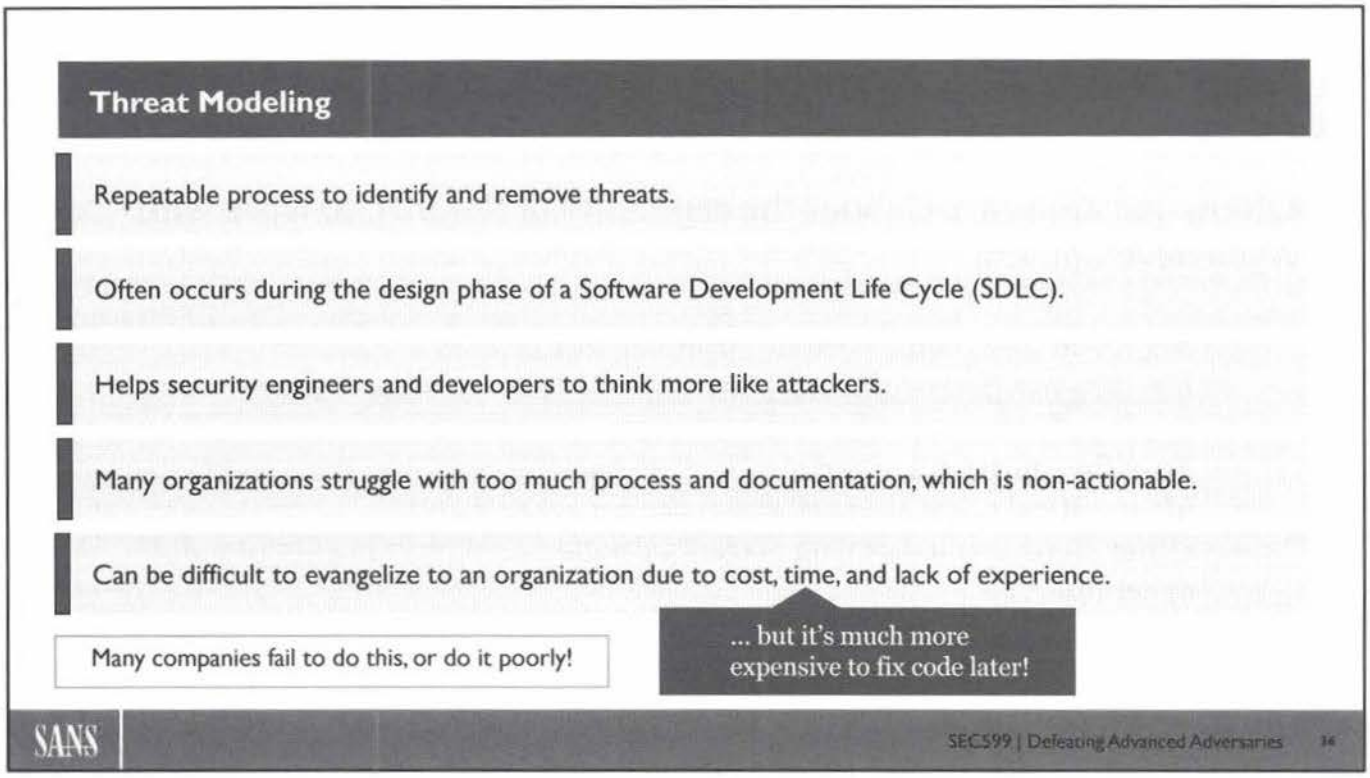
This is not to say that the framework is not applicable; it is simply saying that the application of the various steps during each phase may have to be customized to meet the needs of the organization and the security requirements.

Again, the biggest selling point is that a properly implemented SDL should result in quantifiable savings. It should make for an efficient development process, and there should be a noticeable change and decrease in code fixes. The term return on security investment (ROSI) is often a helpful approach. The general idea is that by spending time and money doing something to reduce or avoid a potential or existing risk will prevent a future loss that would likely be greater than the cost of mitigating the risk.




**Agile Development with the SDL**

Agile development is a development process that is highly utilized and often difficult to implement. It is often seen as an inhibitor to creativity by many developers who have not successfully implemented the process and changed from models such as the waterfall model. Microsoft set up a specific application of the SDL to agile development methods that can be viewed at http://www.microsoft.com/security/sdl/discover/sdlagile.aspx. It maps specific portions and steps of the standard SDL previously covered to different development phases using the agile approach. Every agile sprint receives the most critical steps of the SDL based on the biggest areas of concern. The most important tools are run, threat modeling is performed, and code review is performed, as well as various other security reviews. A sprint is typically several weeks long and a fast-paced subset of development for the overall product. Applying all phases of the SDL to every sprint is not actionable. The other areas of the SDL that are not applied during every sprint can be applied at project initiation, such as those relative to the requirements and design or during bucket practices that occur at set intervals.

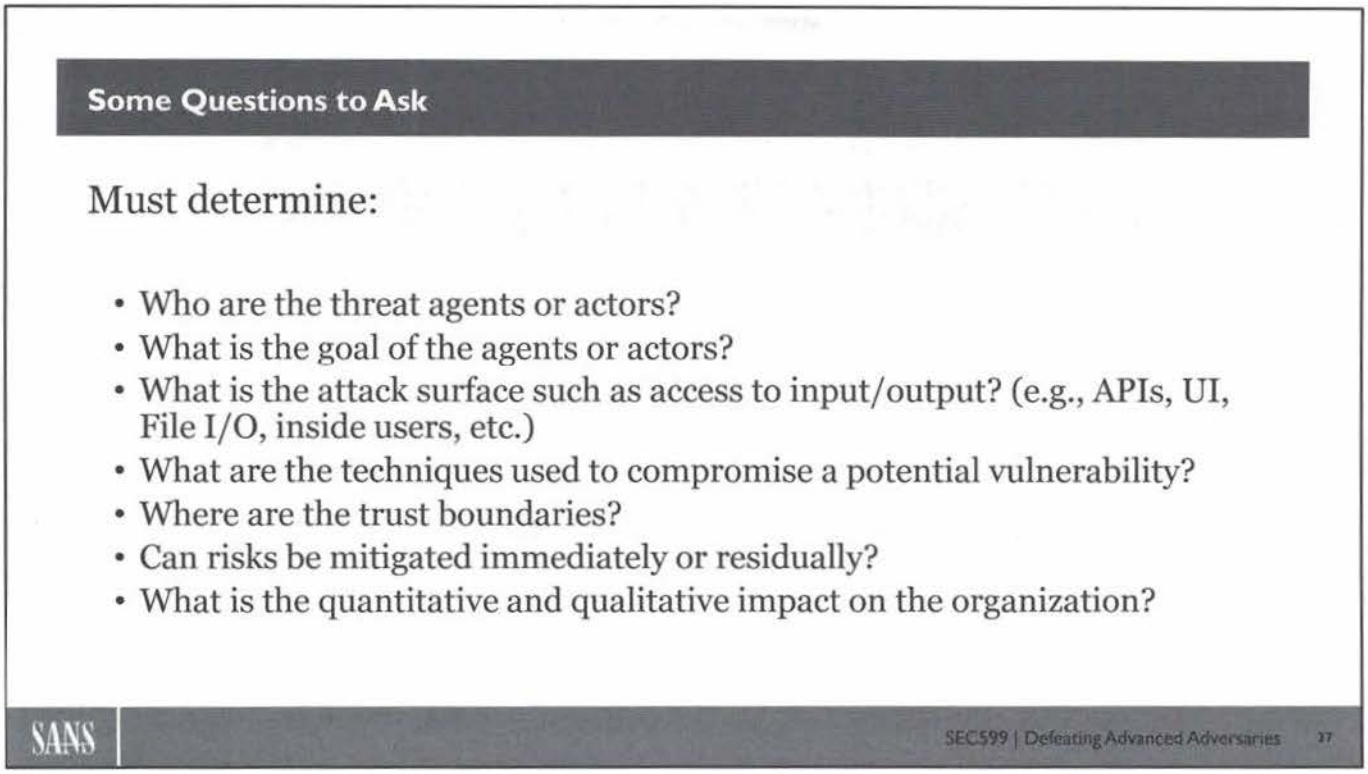



**Threat Modeling**

Threat modeling is an extremely valuable resource if implemented properly. Think about the cost associated with reviewing and fixing production code, or even code that has not been published yet, when a significant finding is found. Often times a vulnerability may be left in the code due to the results of a risk assessment showing that the cost would be greater to fix the bug compared to the impact to the organization if it was discovered and exploited. Regardless of that assessment and justification, it would clearly be more desirable if that bug had never been introduced in the first place. This is where threat modeling can help.

Threat modeling is easy to talk about and hard to implement into an actionable process. It used to be that few developers and security professionals knew exactly what threat modeling was and how it was to be implemented. With the help of various organizations such as Microsoft, Cigital, and OWASP, threat modeling has been made more actionable and dynamic. Similar to that of Microsoft's SDL, it is not a process that can just be implemented with perfect results. It takes time and effort, with much training and practice. Threat modeling is commonly performed as part of the design phase in the development process. Once the low-level diagrams are available—showing all of the data flows and processes—it is much easier to look at the attack surface and point out potential vulnerabilities. The goal is to make an actionable, repeatable process in the design phase of the Software Development Life Cycle (SDLC) to prevent vulnerabilities from being introduced into the code or overall architecture.

Many organizations get too focused and overwhelmed with documentation and process. This becomes nonactionable and slows down the development process. It is better to simplify the threat modeling process and focus on the biggest areas of concern, rather than try to accomplish too much at once and lose support for the initiative. It must also not impede the developer's ability to be creative, especially in product-based companies. This goes for the overall SDL process as well. Similarly to selling the SDL process, it can be difficult for some organizations to gain support for threat modeling. Demonstrating the process, evangelizing it, showing other companies who is using the process, and starting small can help. You must remember to map technical risks into business terms to ensure the request has teeth.

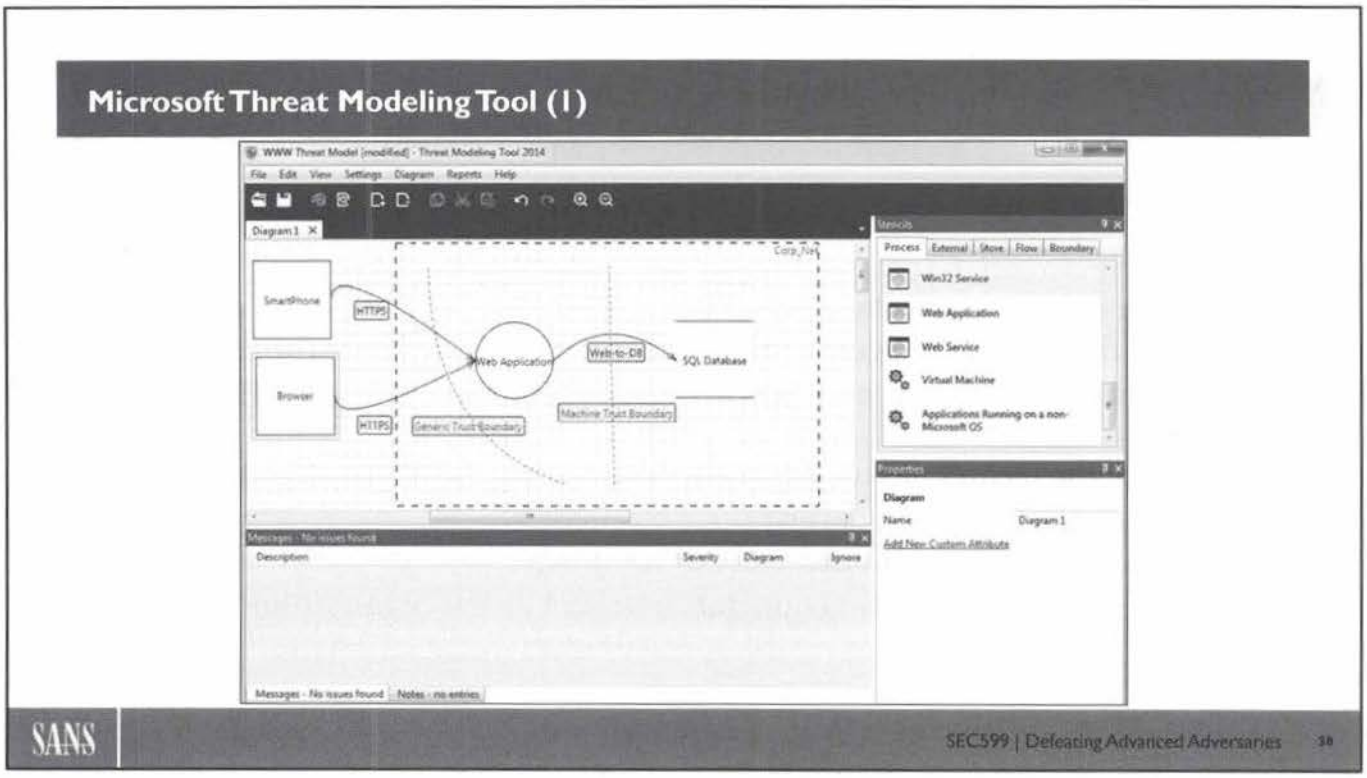



**Some Questions to Ask**

Once you have the design to which you want to apply threat modeling and you ensure it is sufficiently low level, there are many questions to start asking. There are various publicly available threat models such as STRIDE from Microsoft, as well as additional risk assessment models such as Microsoft's DREAD, the Department of Homeland Security's (DHS) Common Vulnerability Scoring System (CVSS), Carnegie Mellon's OCTAVE, TRIKE by Brenda Larcom and Eleanor Saitta, and many others.

We want to know about the threat agents or actors. These could be inside users with privileged access, malicious users from home on their computers or over phones, malicious software, jailbroken smartphones, and countless other threats. We want to understand their potential goals, such as harvesting credit card numbers, denial of service, and intellectual property theft. What is their attack surface? Perhaps they are able to communicate with our frontend web servers with no authentication and then have additional opportunities with authentication. Is authentication assumed once initially authenticated? What else is exposed? DNS servers, mail servers, etc. Do we have store branches? Is social engineering a possibility? How about more complex attacks like communications occurring from inside a trust boundary? Get creative. What techniques are used to exploit the attack surface and potential vulnerabilities identified? According to OWASP, the attack surfaces include all data flows in and out of an application, the code that protects these flows, the data elements involved, and the code that protects those elements. Check out the following cheat sheet for some tips from OWASP: https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet.

Can these risks be mitigated through existing controls? Is it possible to fix them with code? Sometimes the vulnerabilities identified during threat modeling prove challenging to fix and the fixes do not always come from code changes. You must always assume that communications coming from outside of a trust boundary could be malicious. What happens if someone breaks out of the security controls enforced by an embedded device? They can now potentially reverse engineer a mobile application, proxy the communication, and circumvent security restrictions. As we do with any type of risk assessment, we must determine the quantitative and qualitative impact on the organization. How bad could it be? How much would it cost? What is the likelihood?




**Microsoft Threat Modeling Tool (1)**

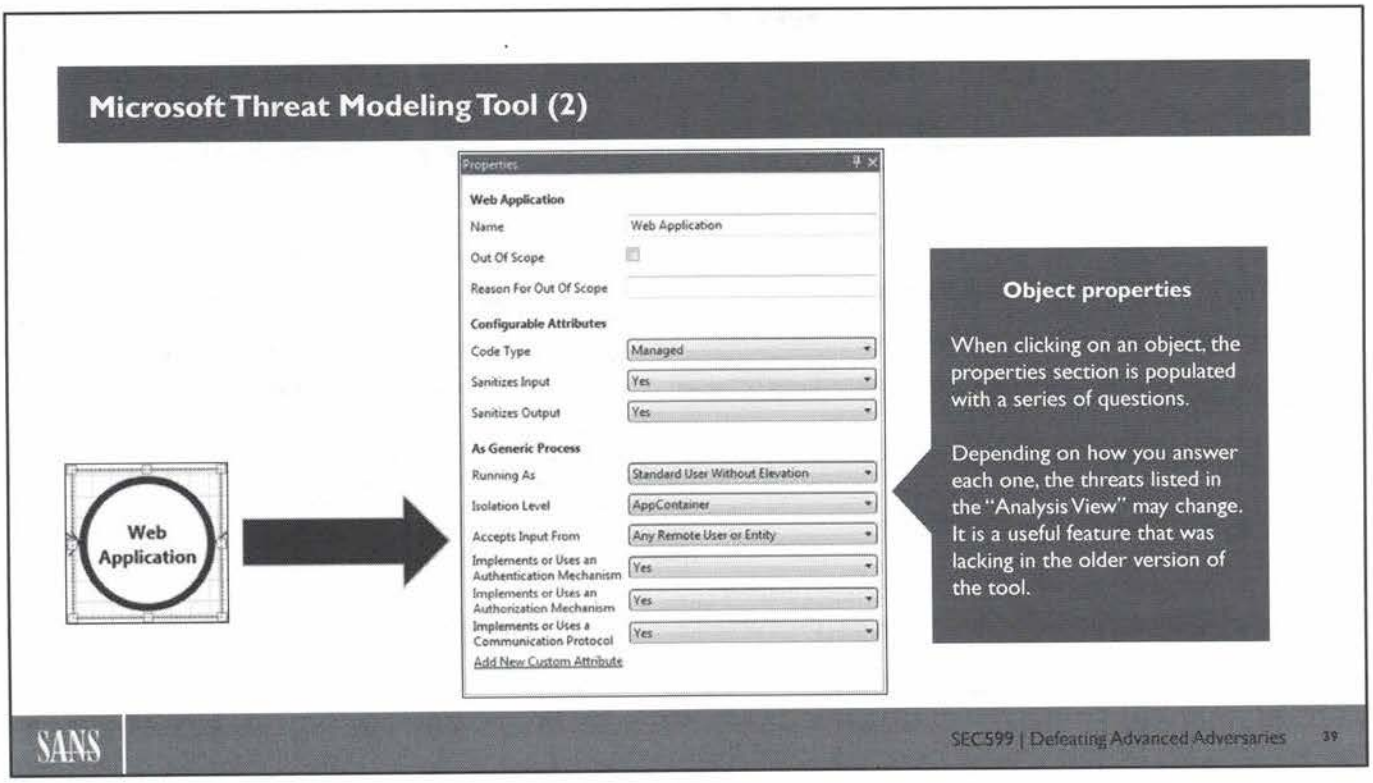
Microsoft released a free tool simply called the Threat Modeling Tool. You can download the Microsoft Threat Modeling Tool version 2014 at http://www.microsoft.com/en-us/download/details.aspx?id=42518 General information about the tool can be found here: http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx. Microsoft also released a great card game called the "Elevation of Privilege Card Game" to practice threat modeling against your designs. It is available at http://www.microsoft.com/security/sdl/adopt/eop.aspx.

With the Threat Modeling Tool, you can draw your designs and have an automated tool get you started on asking the right questions. It used to require MS Visio; however, with the new version released in March 2014, Visio is no longer required. The initial screen, as shown on the slide, is the "Design View." This is where you actually draw out your designs to the whiteboard and make all of the relevant connections and data flows. One nice thing is that the "Messages" area on the bottom, which will let you know if you have likely missed a data flow. The example drawn on the slide is that of a simple network communication. The red hyphenated lines indicate a trust boundary where increased attention should be placed. From within a trust boundary, data and flows may be implicitly trusted, as where communications coming into or leaving a trust boundary should be more aggressively scrutinized.

There are other free tools available on threat modeling, such as Seasponge from the Mozilla Winter of Security 2014:

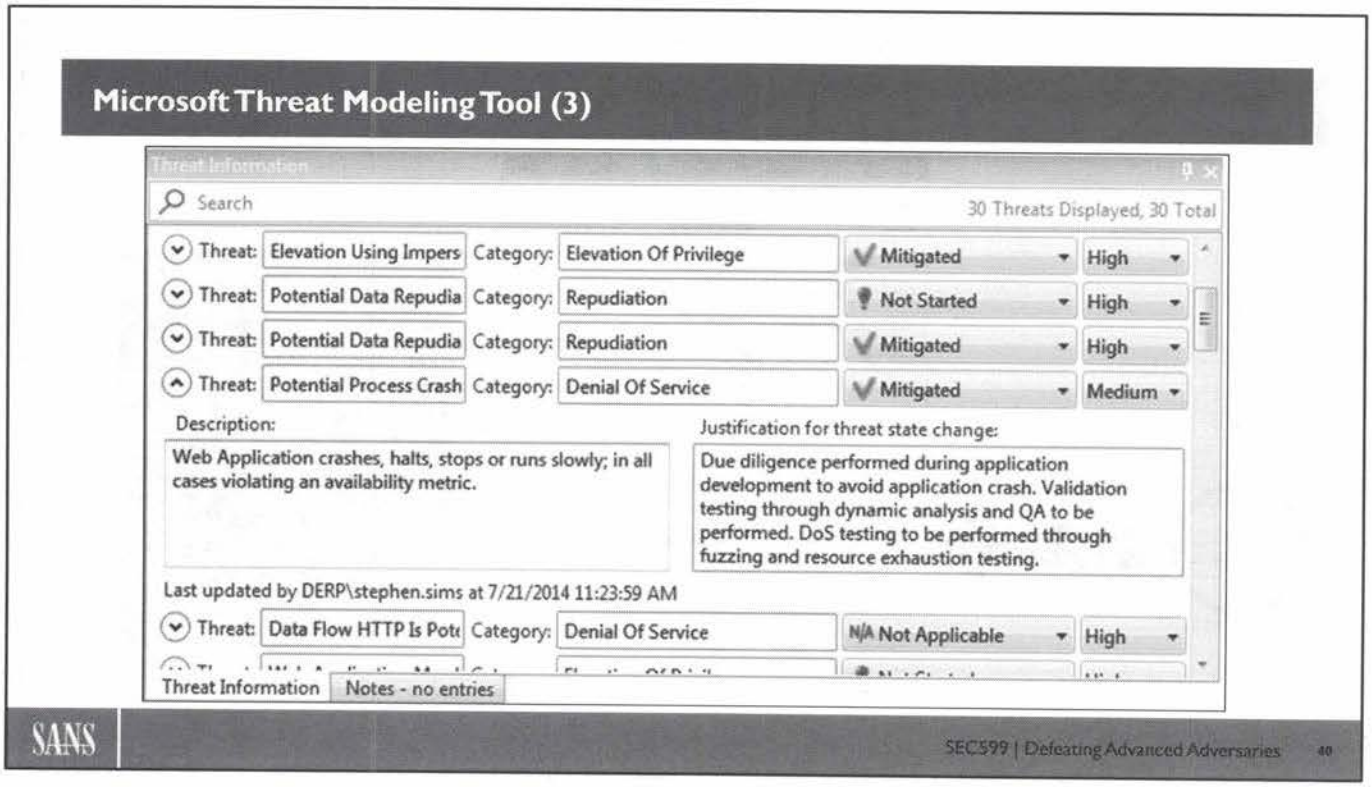
https://air.mozilla.org/mozilla-winter-of-security-seasponge-a-tool-for-easy-threat-modeling/

https://github.com/mozilla/seasponge




**Microsoft Threat Modeling Tool (2)**

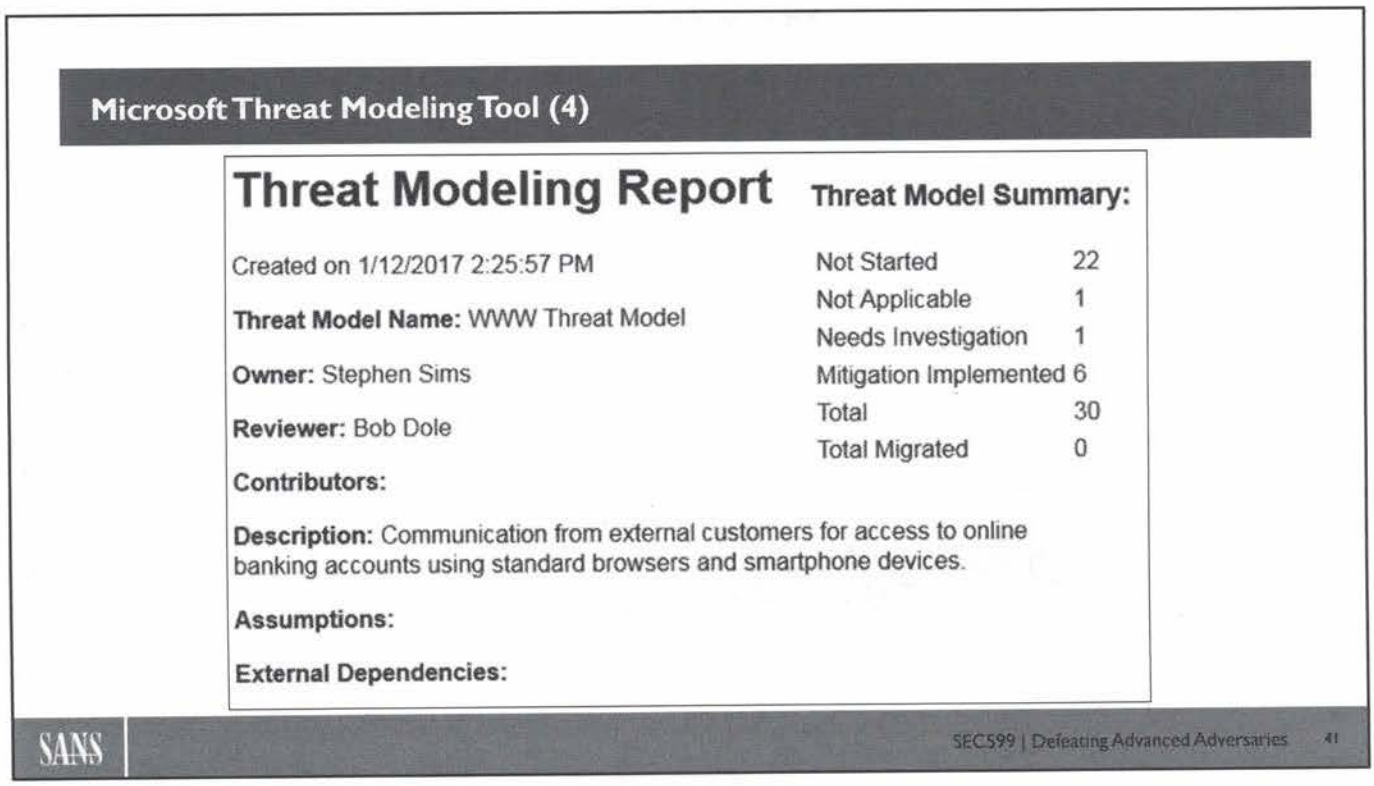
On this slide is a screen capture of the "Properties" section of the Threat Modeling Tool. When you click on certain types of objects, this region will be populated with a series of questions. Depending on how you answer each one, the threats listed in the "Analysis View" may change. It is a useful feature that was lacking in the older version of the tool.




**Microsoft Threat Modeling Tool (3)**

This slide shows the "Analysis View" screen. Once you have drafted your design into the design window, click on "View, Analysis View" from the ribbon bar to see what threats have been identified by the Threat Modeling Tool. It is designed to get you thinking about potential threats and add some automation for developers who may not be security experts. That being said, the tool does a great job at asking the initial questions that should be asked or making simple comments such as, "Web Application crashes, halts, stops or runs slowly; in all cases violating an availability metric." This is an example of a topic that may not be brought up without the help of the tool. Not all of them will apply to each flow and they can be removed if appropriate.

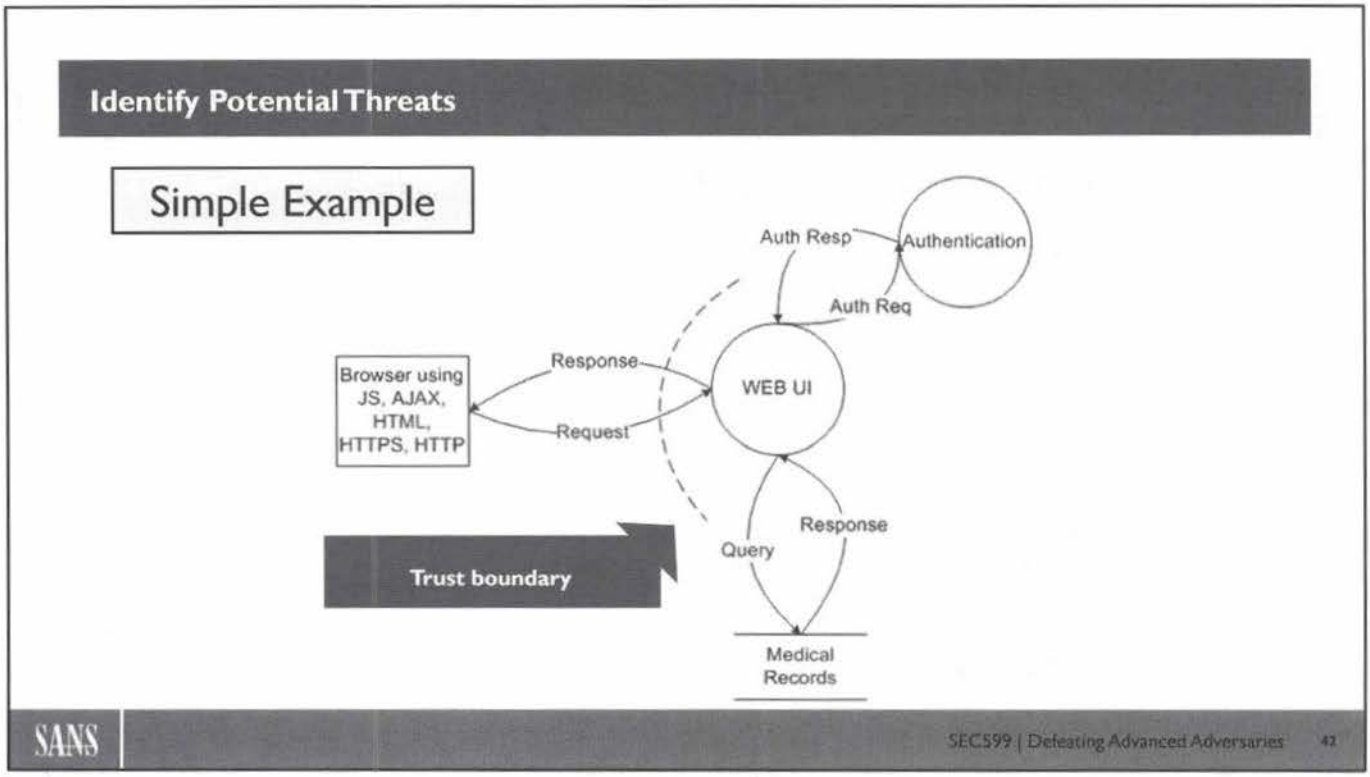
As seen on the slide, there is a Threat and Category. Following that are drop-down boxes showing the status of the risk item and the qualitative rating. On the left of each threat is a drop-down arrow that expands the description of the threat. In the example shown, you can see that the "Justification for threat state change" area on the right is populated with user-supplied content.




**Microsoft Threat Modeling Tool (4)**

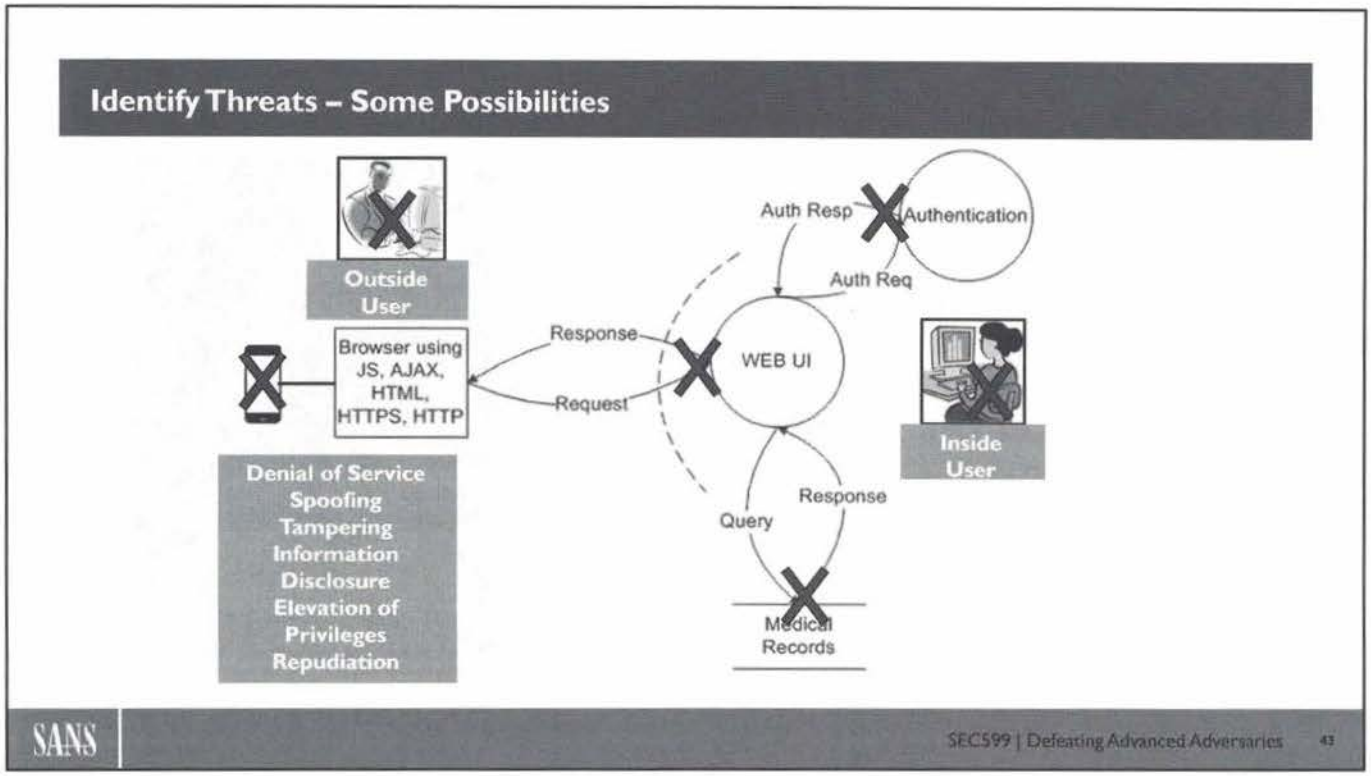
When clicking on "Reports" from the ribbon bar, you can select the option to generate a full report of the threat model. On the slide is a snippet of that report. Note: Pieces of the report were moved around to fit on the slide. As you can see, the information about the threat model author, description, assumptions, and dependencies are shown. A summary of the number of threats and the ones still needing to be triaged are also shown. Below this section in the HTML-generated document are all of the threats listed associated with each device, trust, store, or data flow shown.

A zip file is available from Microsoft called "Threat Modeling Tool 2014 Principles.zip." It contains a video, sample threat models, and documentation. It is available for download at http://aka.ms/By12as.




**Identify Potential Threats**

On this slide is a very simple network diagram as created with the older version of the Microsoft Threat Modeling Tool. This simplified example is a good place to start when practicing threat modeling. Take a few minutes to identify potential threats. Note the trust boundary marked by the curved, hyphenated line. Trust boundaries require special attention and often influence what components will be fuzzed. On the next slide are some of the potential areas of concerns that should be addressed prior to implementing the design.

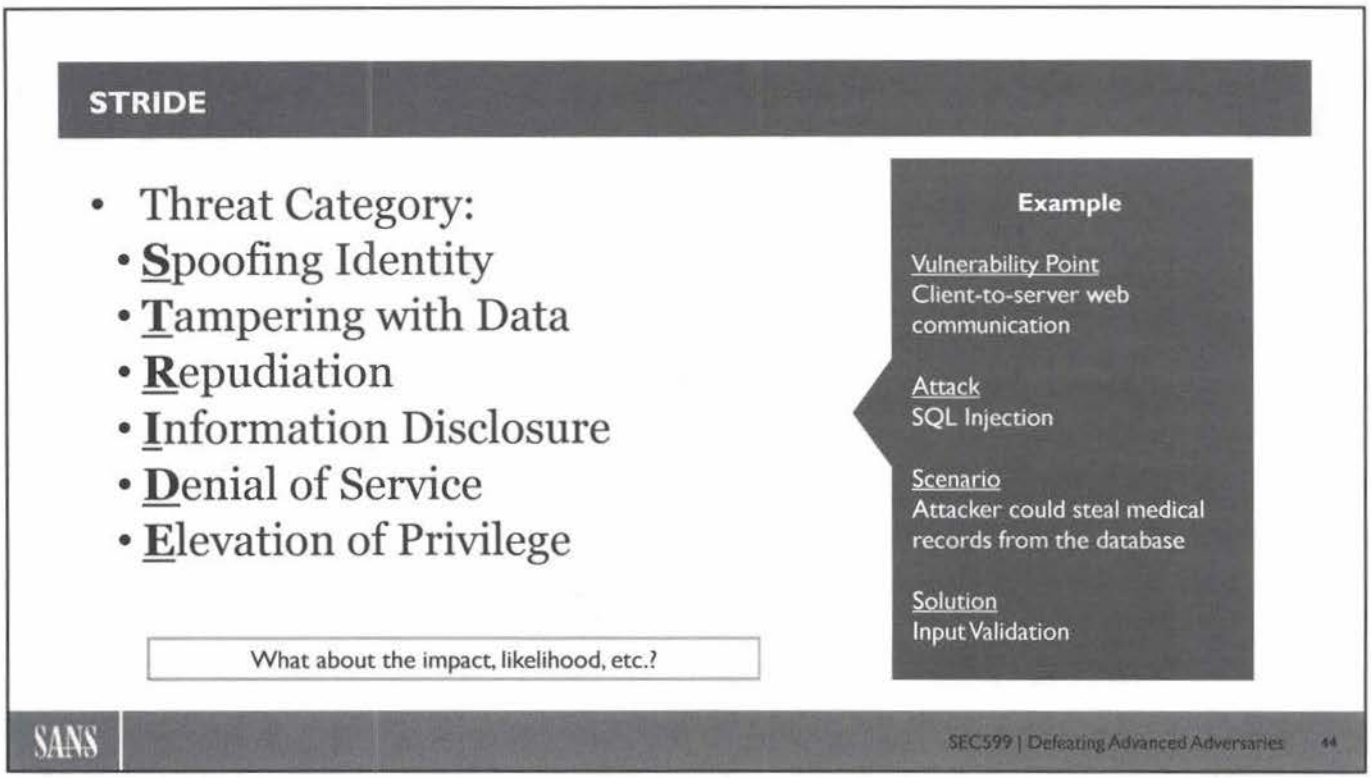



**Identify Threats – Some Possibilities**

On this slide are some potential threats and vulnerability spots that must be addressed. Some examples of attack categories are Denial of Service (DoS), spoofing, tampering, information disclosure, elevation of privilege, repudiation, and many others. Not all apply to each threat or vulnerability, and we can rule them out as they are addressed. The outside user could be on a personal computer, a smartphone, a kiosk in a store branch, and other possibilities. This is where it is important to think like an attacker. What about the scenario in which your company creates a smartphone application that should be protected by the controls included with an iPhone. Let us say that the attacker jailbreaks the iPhone and is able to circumvent all controls, install his own software, reverse engineer, and learn more about your smartphone application, proxy connection requests, etc. Does this change the typical attack surface? It sure does!

Again, it is easy to start with high-level designs when it comes to threat modeling, but the real power comes in when you get into low-level application designs and data flows. It is at the design stage during the SDLC that you can help prevent bugs or design flaws from being introduced by threat modeling. The more this process can be automated, the more likely it is to be adopted. We cannot expect that all of our developers will become security experts overnight, and if this can be rolled into the development process as seamlessly as possible, our chances of success increase.

Are there any missing trust boundaries that stick out? You may have noticed one should be placed between the Web UI and the "Medical Records" data store, as well as possibly between the Web UI and the Authentication.

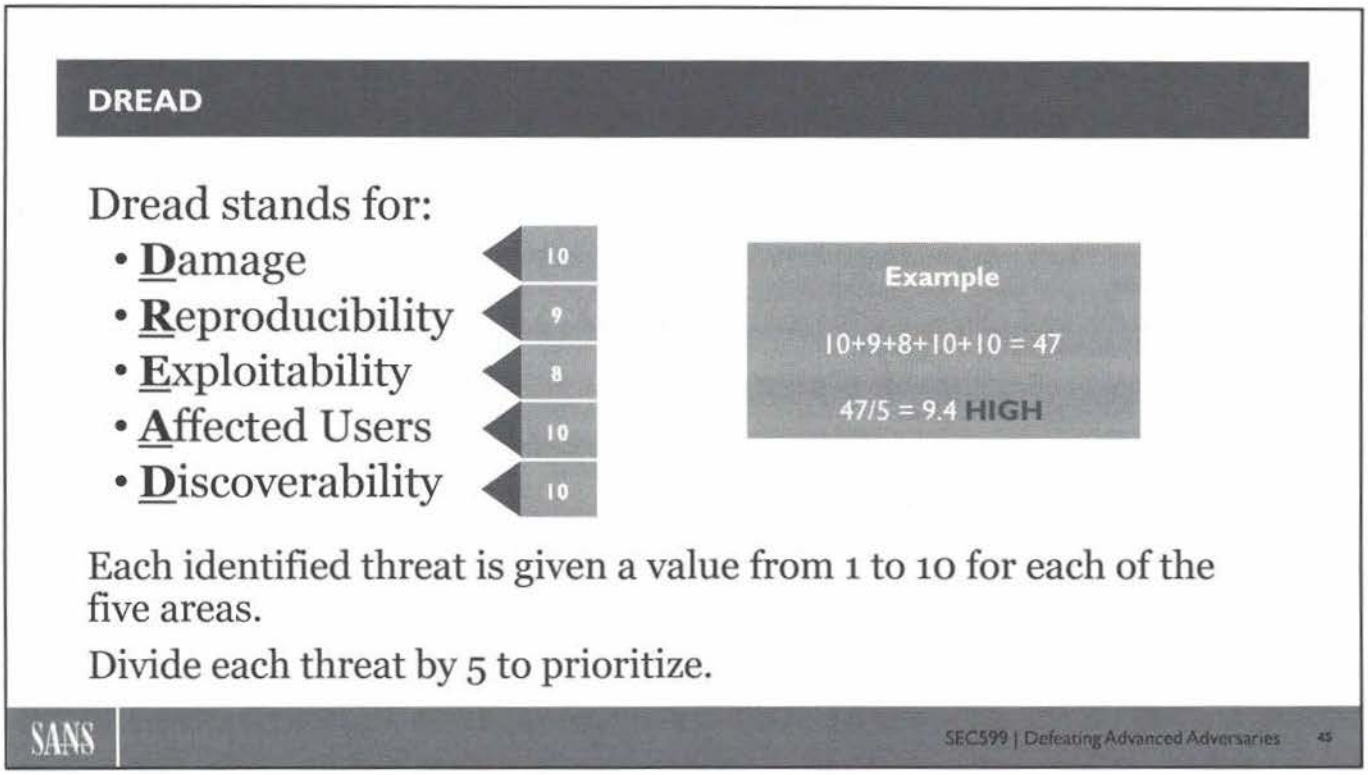



**STRIDE**

The Microsoft Threat Modeling Tool is based on the STRIDE threat model. As previously mentioned, there are quite a few threat models made publicly available by various organizations. The Microsoft STRIDE threat model is available at http://msdn.microsoft.com/en-us/library/ee823878%28v=cs.20%29.aspx.

STRIDE is an acronym that stands for "**S**poofing Identity, **T**ampering with Data, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege." Each of these is a category of threats that should be well known to all of us by this point in our careers. Under each of these threat categories are various attacks, which the Threat Modeling Tool details. The model would have you identify a potential vulnerability point within a design, such as data coming from a user into a web application. Threats to that vulnerability and the attacks associated with them are then identified, such as cross-site scripting (XSS), parameter tampering, SQL injection, etc. Under each of the potential attack types, you would then document some scenarios that could occur. Finally, some mitigations for each vulnerability can be identified.

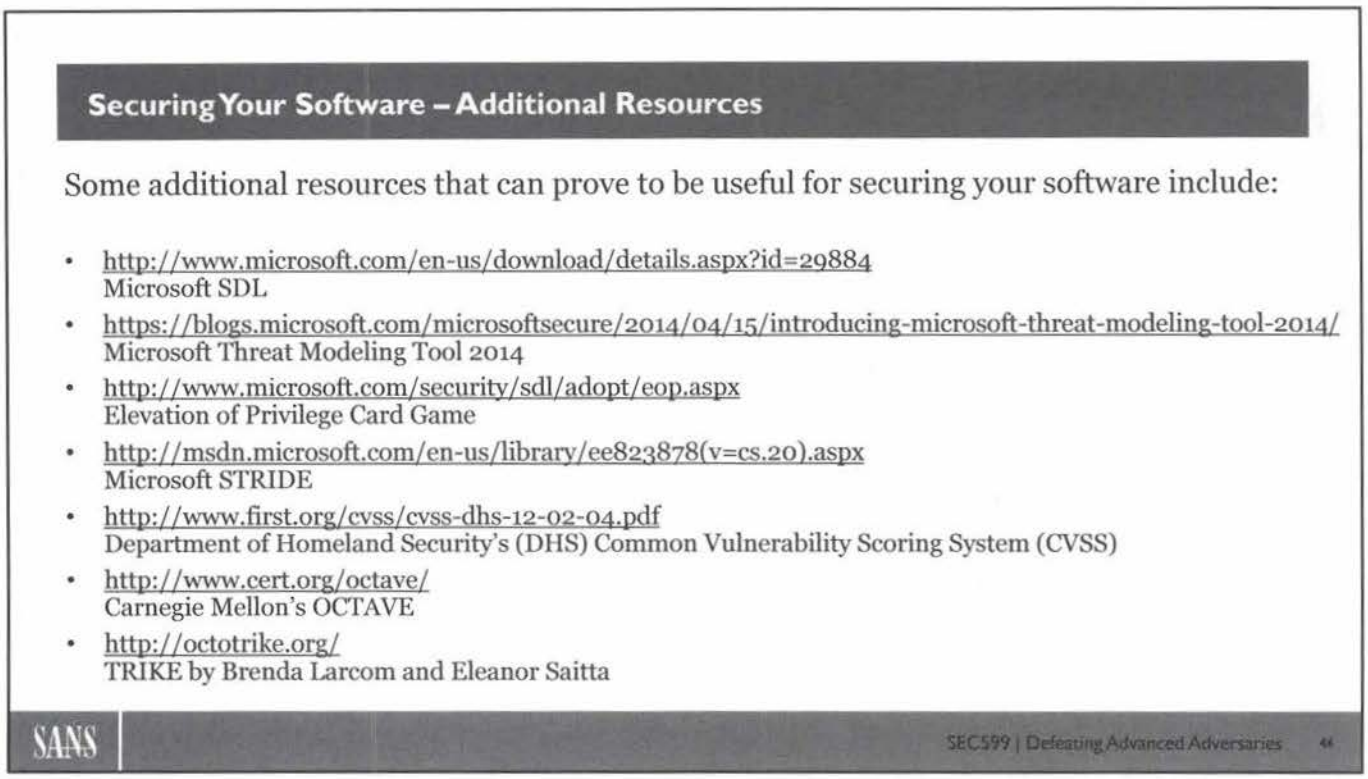
What about the impact of an event, the likelihood, and other risk assessment modeling?




**DREAD**

DREAD is a multidimensional risk calculation model for prioritizing threats. Microsoft documentation on DREAD can be found at http://msdn.microsoft.com/en-us/library/ff648644.aspx.

The five areas applied to each threat include **D**amage, **R**eproducibility, **E**xploitability, **A**ffected Users, and **D**iscoverability. Damage can be compared to the impact a successful attack would have on the organization. A compromised database containing a million patient records in the worst-case scenario would be a grave impact and as such, we assign it a 10. The reproducibility pertains to the likelihood that the attack is successful and reproducible. Once a SQL injection attack is identified, it is typically easy to reproduce with success. We gave this one a 9. The exploitability pertains to the difficulty in pulling off the attack successfully. Is it a well-known attack with lots of tools and help, or is it obscure and difficult? We gave this one an 8. Affected users pertain again to the impact. In our scenario, a million patients are affected and as such we give this one a 10. Finally, we have discoverability that pertains to the likelihood that someone will find out about the vulnerability. In our example, we've assigned this one a 10, as SQL injection vulnerabilities are often easy to spot. Each organization would apply its own ratings to this threat. When we add up each of the five areas, we get 47. We divide this number by 5, representing the 5 areas in DREAD, and we come to our overall rating of 9.4, which can be considered high. We would likely want to address this threat with priority.




**Securing Your Software – Additional Resources**
Some additional resources that can prove to be useful for securing your own software include:

http://www.microsoft.com/en-us/download/details.aspx?id=29884
Microsoft SDL

https://blogs.microsoft.com/microsoftsecure/2014/04/15/introducing-microsoft-threat-modeling-tool-2014/
Microsoft Threat Modeling Tool 2014

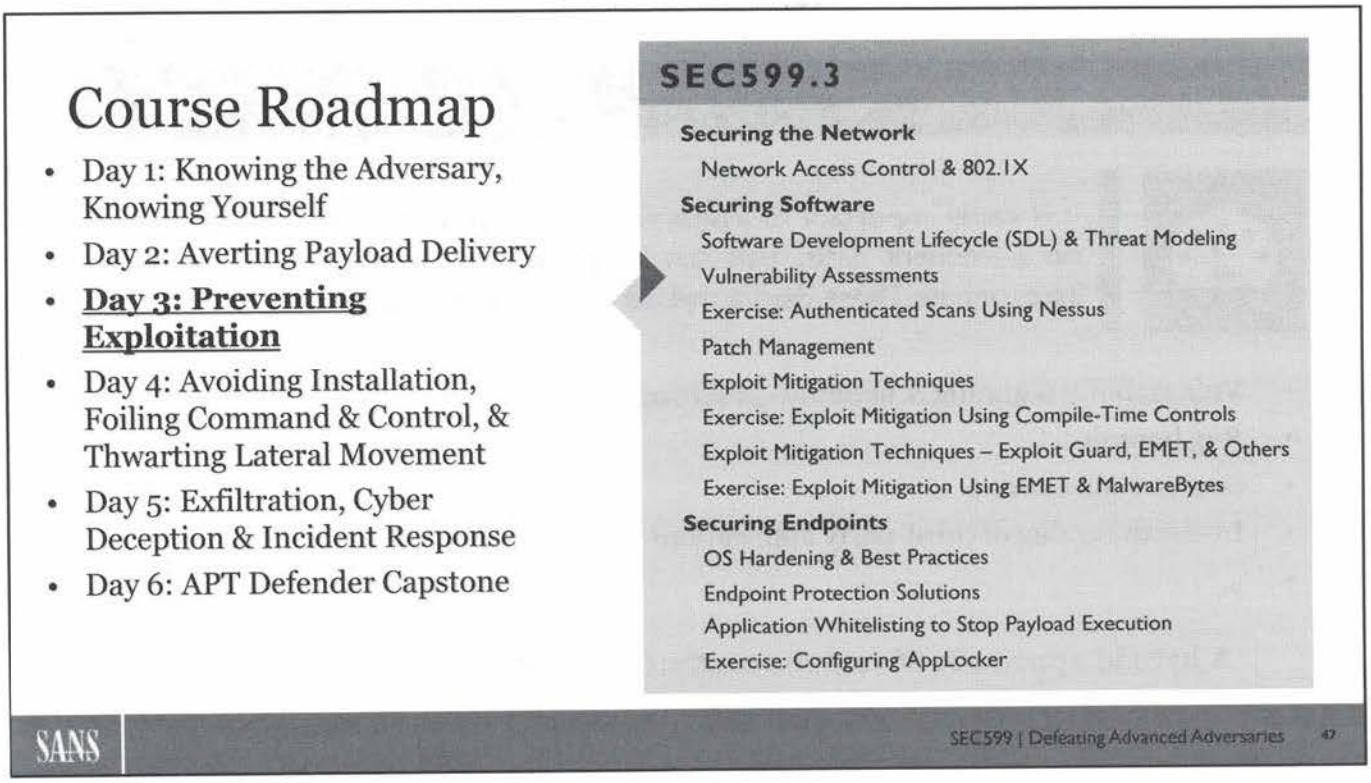
http://www.microsoft.com/security/sdl/adopt/eop.aspx
Elevation of Privilege Card Game

http://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx
Microsoft STRIDE

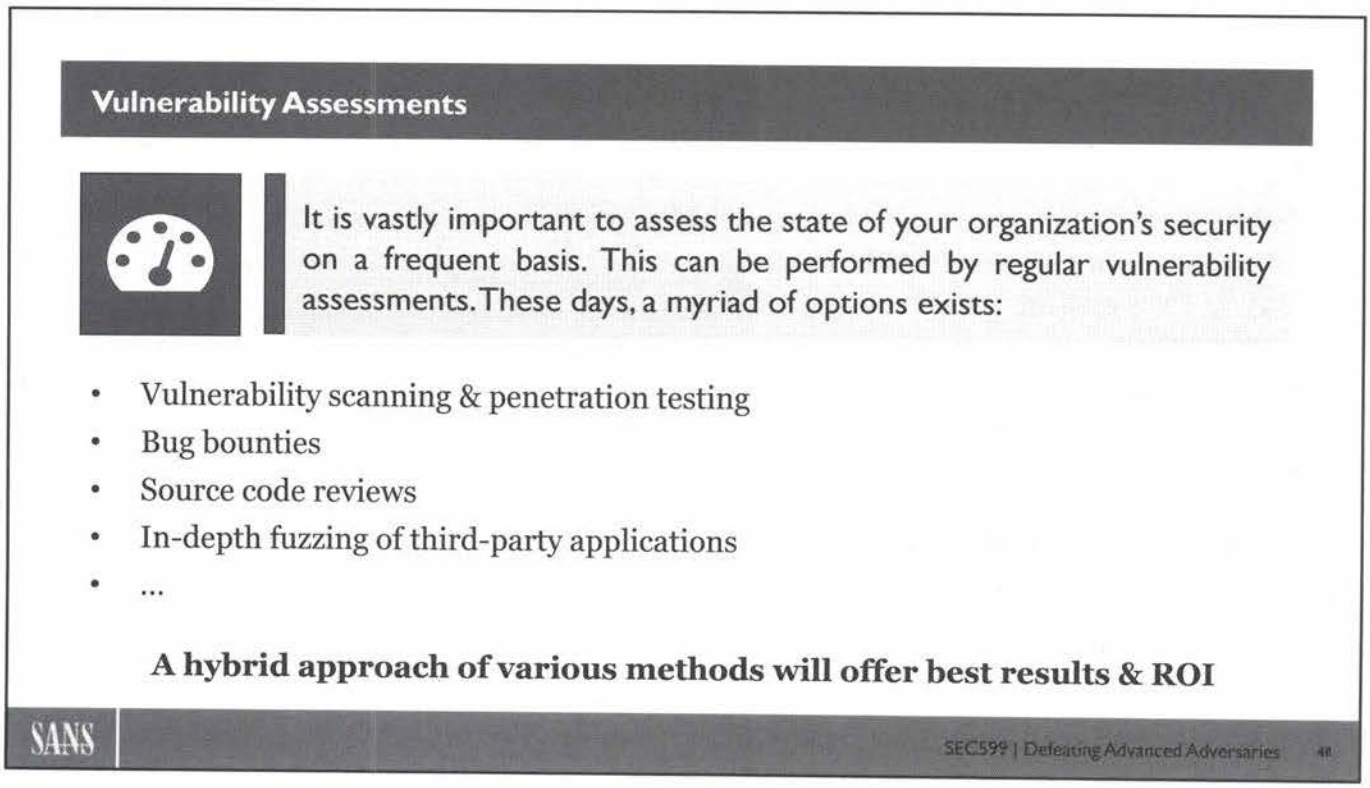
http://www.first.org/cvss/cvss-dhs-12-02-04.pdf
Department of Homeland Security's (DHS) Common Vulnerability Scoring System (CVSS)

http://www.cert.org/octave/
Carnegie Mellon's OCTAVE

http://octotrike.org/
TRIKE by Brenda Larcom and Eleanor Saitta




This page intentionally left blank.

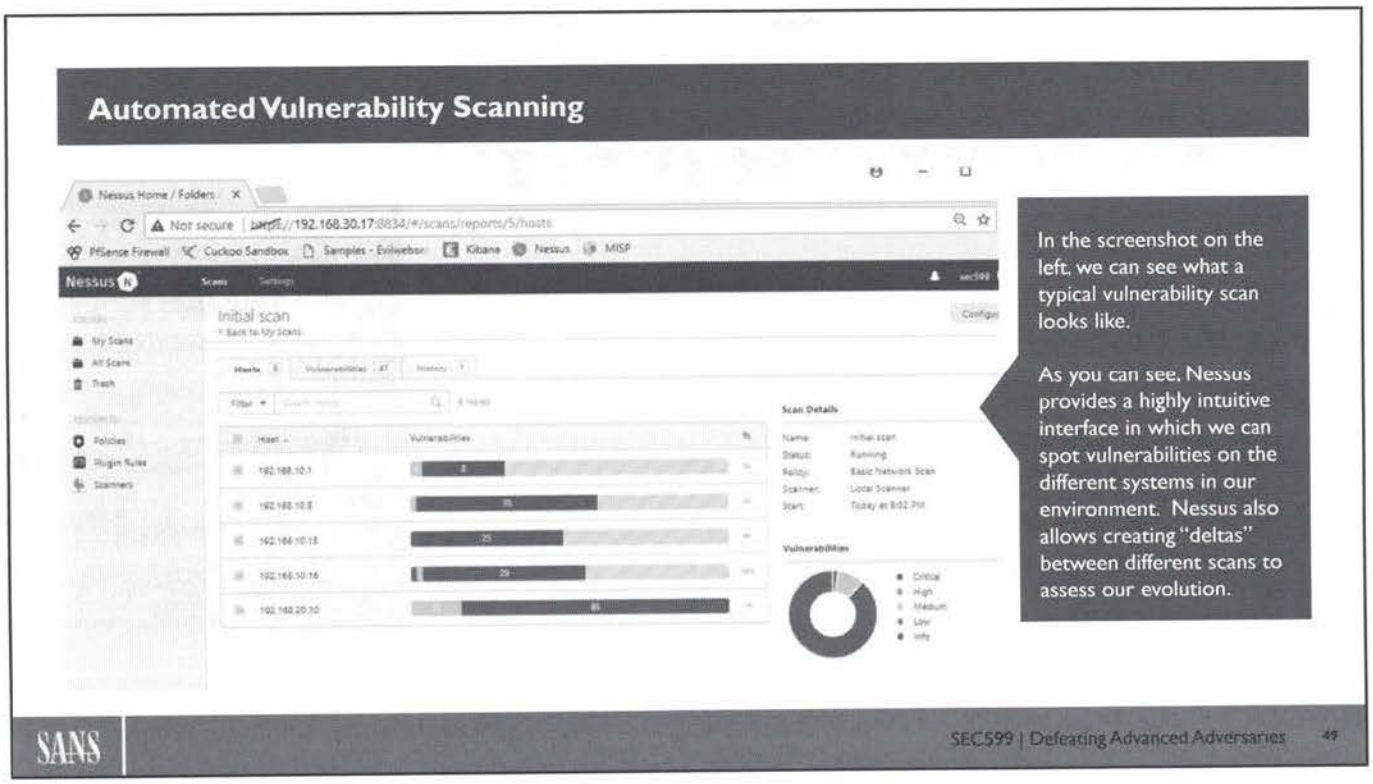



**Vulnerability Assessments**
It is vastly important to assess the state of your organization's security on a frequent basis. This can be performed by regular vulnerability assessments. Due to the state of the current cyber security landscape, vulnerability assessments have become the norm and the vast majority of organizations are combining different techniques to understand their cyber risk exposure. Some of the more popular ones out there include:

- Vulnerability scanning & penetration testing
- Bug bounties
- Source code reviews
- In-depth fuzzing of third-party applications (not common)

So, what approach is best? In reality, there is no silver bullet… A hybrid approach of various methods will offer best results & ROI. "Bug Bounties" are currently on the rise and an increasingly large number of organizations is rewarding security researchers with bounties for submitted bugs. Does that mean mandated penetration testing or source reviews are on the downfall? No, they are still highly useful in a structured SDLC approach, where testing is performed continuously throughout the SDLC, after which bug bounty hunting can take place once the system is in production.

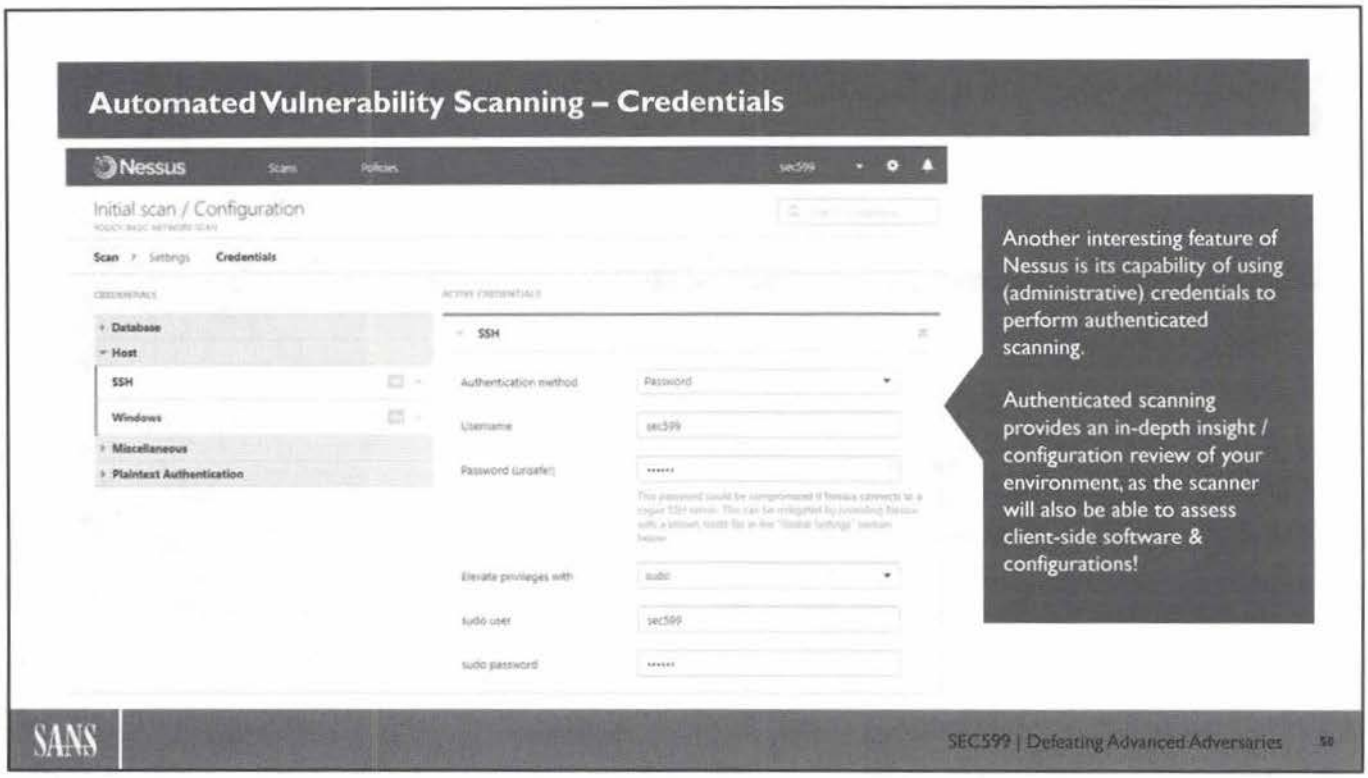
Again, there is no silver bullet; there's just a lot of options to choose from ☺




**Automated Vulnerability Scanning**

In the screenshot on the left, we can see what a typical vulnerability scan looks like. The vulnerability scanner we are using is Nessus, which is a commercial product. Nessus has come a long way. Initially merely a vulnerability scanner, these days it has grown into a full-blown vulnerability management toolkit. As you can see, Nessus provides a highly intuitive interface in which we can spot vulnerabilities on the different systems in our environment. Nessus also allows creating "deltas" between different scans to assess our evolution.

Next to Nessus, different other vulnerability scanners exist with similar functionality. Some of the most popular ones include Rapid7 Nexpose / Insight & Qualys. OpenVAS is an open-source vulnerability scanner that is free of charge. While OpenVAS can be useful, it's important to note it has much fewer plugins as opposed to commercial alternatives and doesn't have the same performance.

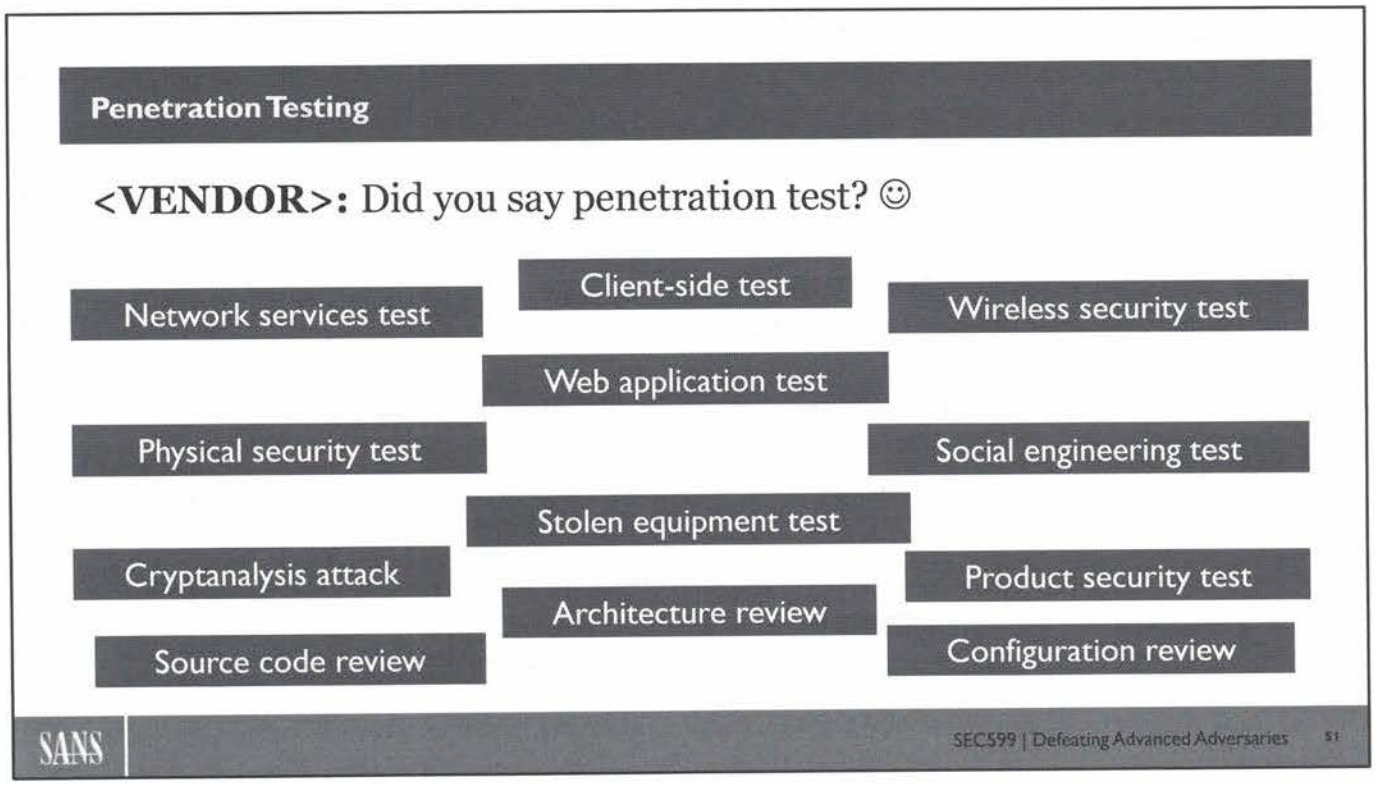



**Automated Vulnerability Scanning – Credentials**

Another interesting feature supported by the majority of vulnerability scanners is authenticated scanning. When doing internal vulnerability scanning, it is highly advisable to enable authenticated scanning instead of unauthenticated scanning:

- Unauthenticated scanning will perform basic network discovery & enumeration ("port scanning"); after, it will run a number of plugins to assess vulnerabilities exposed on available network ports;
- Authenticated scanning will actively connect to the systems (using administrative credentials preferably) and will review the internal configuration of the system. It will thus also be able to identify issues (missing patches or misconfigurations) related to client-side software, which is one of the most dominant attack vectors today.

The slide above illustrates Nessus' configuration, where a credential for a Linux system is being provided.

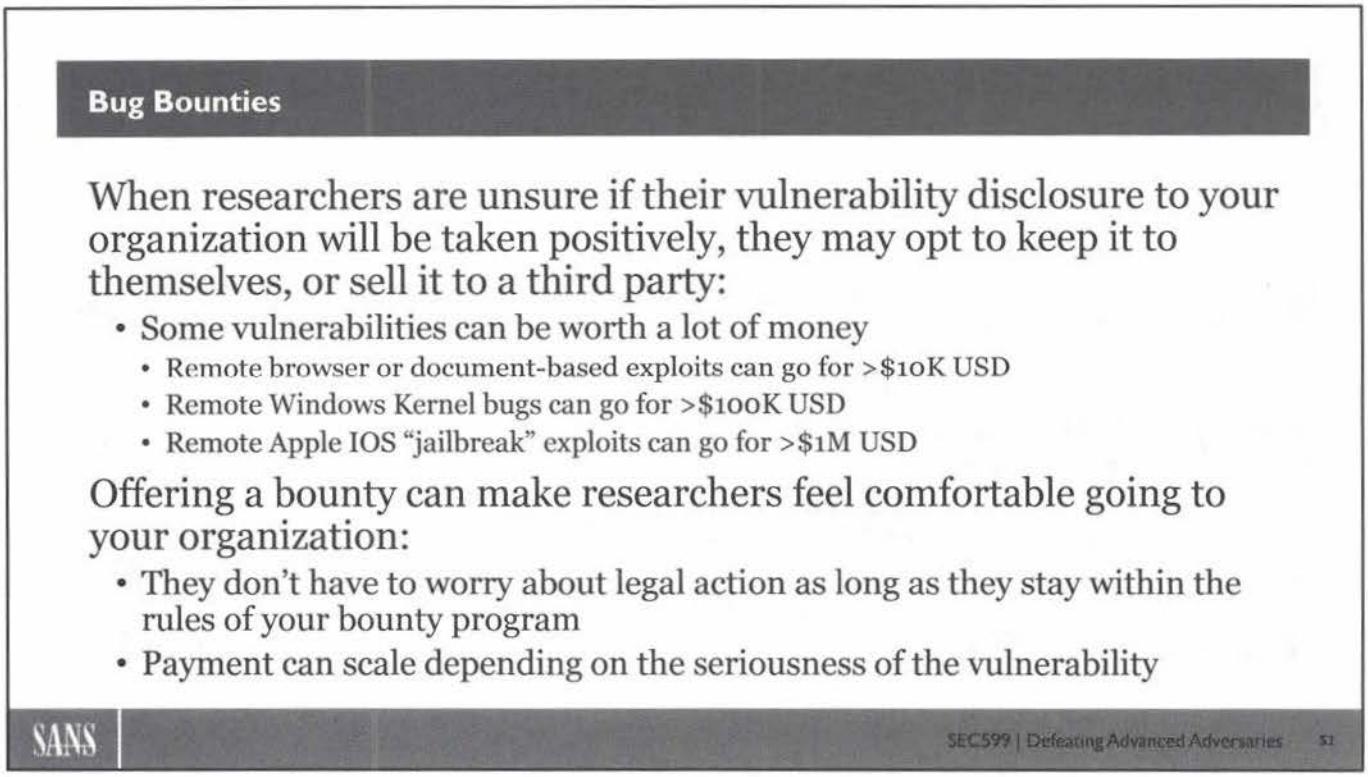



**Penetration Testing**
Penetration testing is a highly mature service in today's cyber security landscape. Many vendors offer a wide variety of services. Even more, many organizations are in-sourcing penetration testing teams to perform continuous assessments. How should you approach it?

- What type of testing should you perform?
- What should the scope be?
- What should the periodicity of testing be?
- Should I go for black-boxing vs white-boxing?
- …

What is best for you? The answer is simple: It depends. We believe strongly in first implementing a number of (basic) security controls, after which they can be tested. For example, it's a waste of effort to launch phishing campaigns if you have never run any security awareness initiatives. We can predict the results up front: if employees have not been educated about security topics they are bound to fall into the trap that is going to be laid out for them. The same is true for secure development; running a source code review against a piece of code that was developed by developers that have never received training or have no "secure coding" standards is bound to result in a report full of findings.

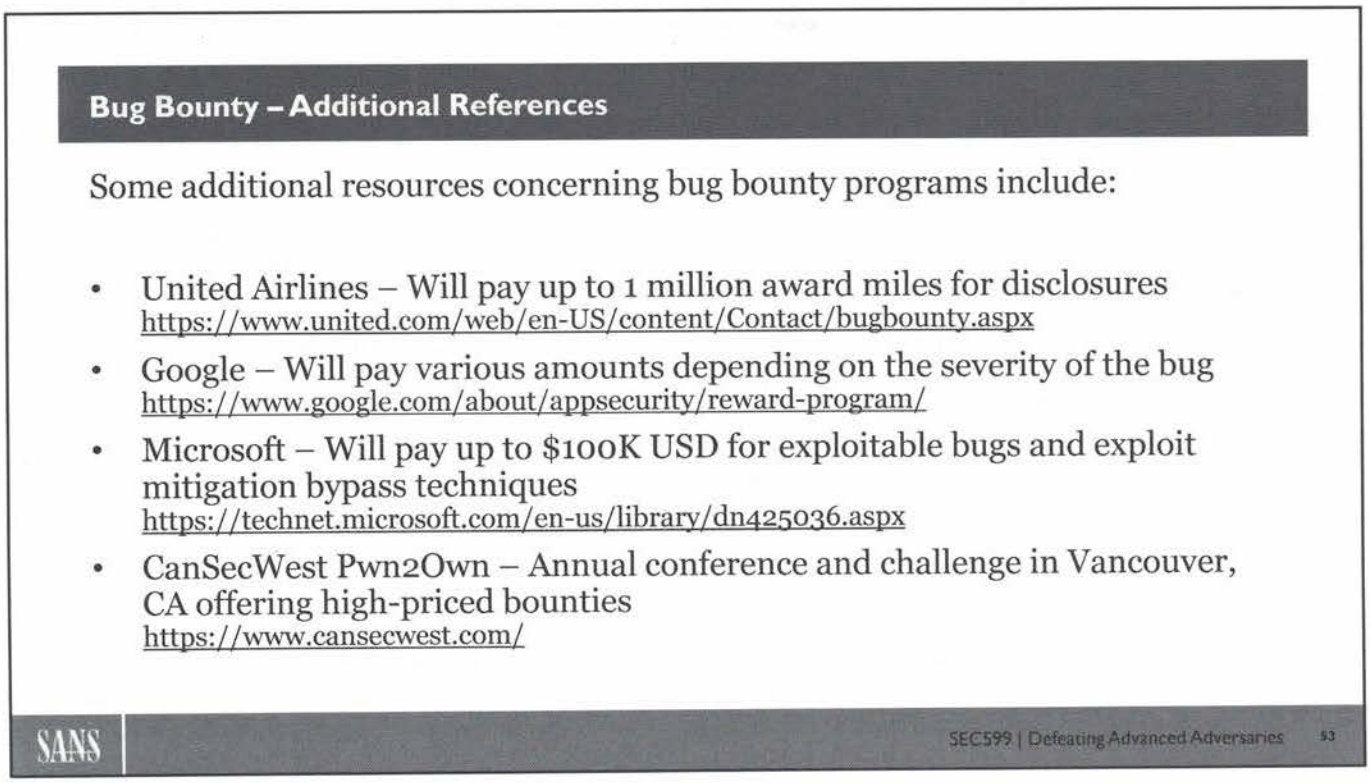
It should, however, be noted that a nice "red-looking" report from a penetration test can often make a point to decision makers and thus "open the gates" for additional cyber security investments.




**Bug Bounties**

The more welcome vendors make those interested in disclosing a vulnerability feel, the more likely they won't sell it to a third-party, release it publicly, or keep it to themselves. If they have to be concerned about potential legal action many will simply avoid the risk. Some vulnerabilities can be worth a lot of money. This author has sold browser-based exploits affecting Internet Explorer for $10K - $20K USD to ethical buyers. Ethical buyers are those who disclose the vulnerability to the affected vendor and do not release details publicly, such as iDefense from Verisign and Tipping Point's Zero Day Initiative (ZDI). Windows remote code execution Kernel bugs can go for over $100K USD. Zerodium paid $1M USD to a group who disclosed an iOS remote jailbreak exploit. See here for details: https://www.zerodium.com/ios9.html Some of the buyers are not as transparent in terms of what they do with a disclosed vulnerability. Some have a customer list that may be seen as questionable by some.

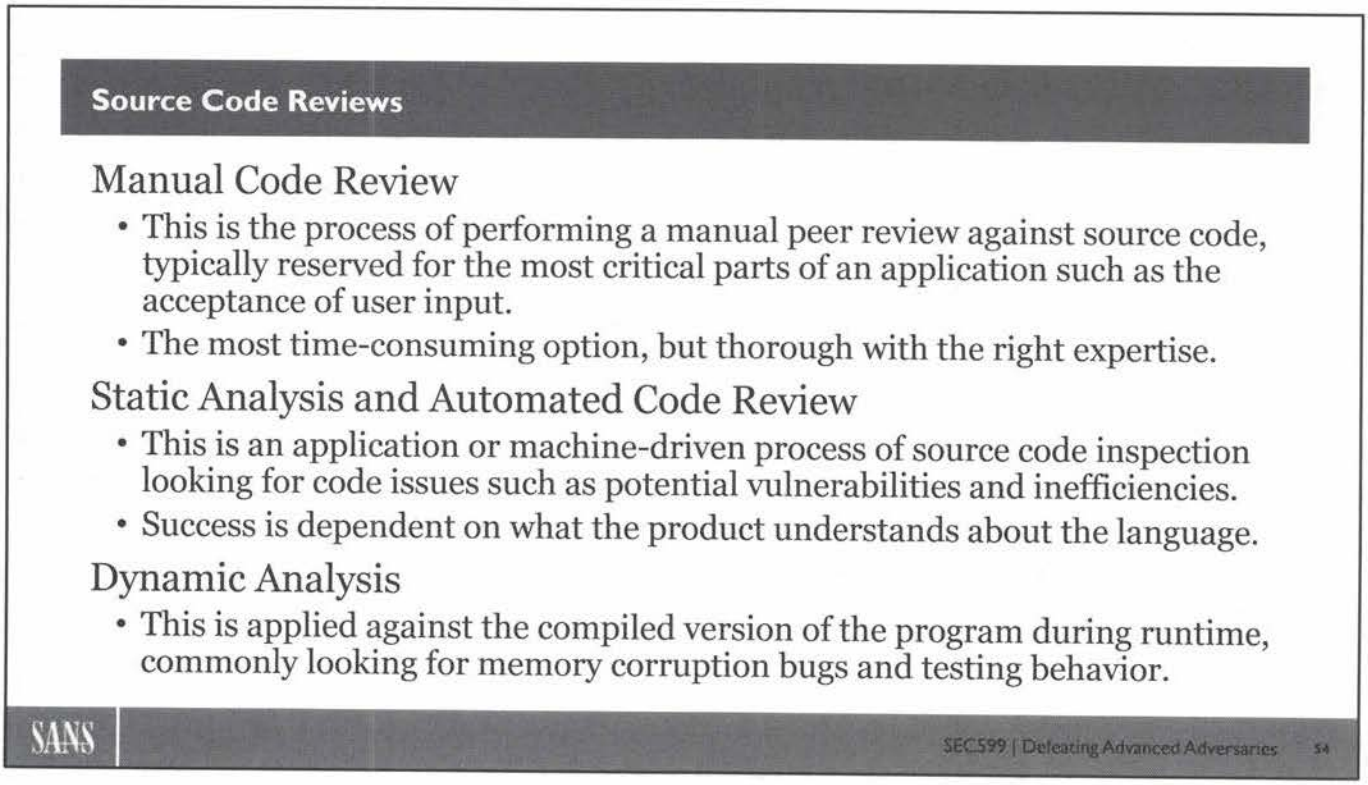
Offering a bug bounty program can help to make researchers interested in disclosing a bug more comfortable and more likely to disclose it to the affected vendor. As long as they follow the rules in the bounty program they should have no reason to fear legal action. The payment should scale based on the severity of the vulnerability. If the bounty is too low, then it is more likely for the disclosure to get taken elsewhere.




**Bug Bounty – Additional References**

There are a large number of vendors who offer different types of bug bounty programs. A few examples are listed on this slide, with some offering bounties such as airline award miles and others offering cash bounties in excess of $100K USD.

- United Airlines – Will pay up to 1 million award miles for disclosures
  https://www.united.com/web/en-US/content/Contact/bugbounty.aspx
- Google – Will pay various amounts depending on the severity of the bug
  https://www.google.com/about/appsecurity/reward-program/
- Microsoft – Will pay up to $100K USD for exploitable bugs and exploit mitigation bypass techniques
  https://technet.microsoft.com/en-us/library/dn425036.aspx
- CanSecWest Pwn2Own – Annual conference and challenge in Vancouver, CA offering high-priced bounties
  https://www.cansecwest.com/

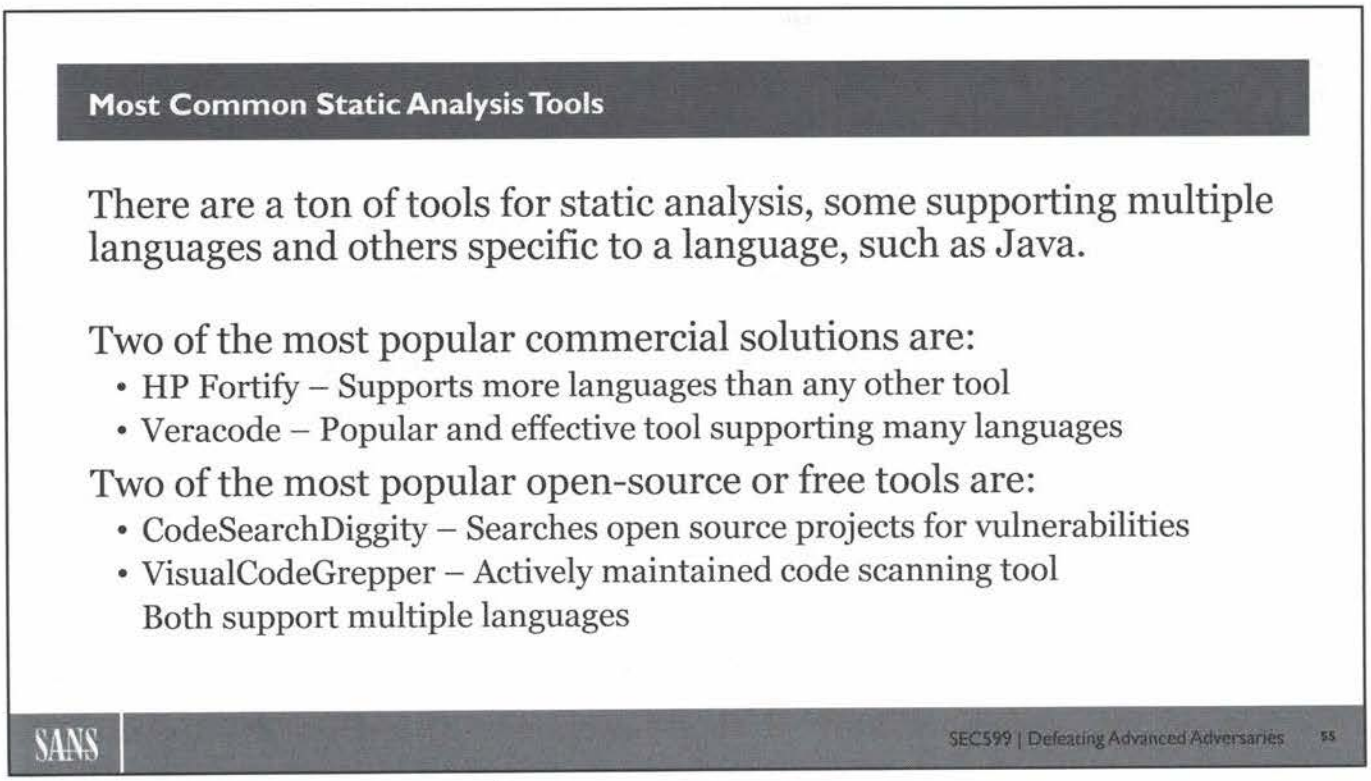



**Source Code Reviews**

There are various types of code reviews that can be performed. Manual code review is often seen as the most thorough but is time-consuming and typically reserved for the most critical parts of an application. This requires someone with the expertise to go line by line through code written by the development team. The quickest option is the use of static analysis tools. Tools such as the Fortify Static Code Analyzer from HP can take an entire project and quickly scan it for easy to spot vulnerabilities, banned functions, and programmatic inefficiencies. The good thing about static analysis tools is the speed at which they can review the code base. One issue is that they often produce a lot of false positives and miss complex bugs. The more they are taught about the supported languages being scanned, the better they can find problems. The term static analysis is also used when disassembling a compiled program for review with a tool such as the Interactive Disassembler (IDA). Dynamic analysis is typically performed during the validation phase of the SDLC once the code has been compiled. Various runtime tools are available such as Microsoft's AppVerifier for unmanaged code. (Unmanaged code refers to low-level languages such as C, C++, and assembly. Managed code refers to languages such as Java, C#, and Ruby.) These tools look for program issues typically related to problems such as heap corruption that are difficult to locate with static analysis. An example of a dynamic analysis tool for managed code is FxCop from Microsoft.

For more information on AppVerifier and FxCop check out the following links:

https://msdn.microsoft.com/en-us/library/ms220948(v=vs.90).aspx
https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx




**Most Common Static Analysis Tools**

If you use Google to search for "static analysis tools" you will get countless results. There are many lists put together by practitioners stating what they claim to be the best free and commercial solutions available. Some of these tools are specific to one or two languages, while others offer support for dozens of languages. There are both commercial tools and open-source or free tools. Two of the most popular commercial solutions in use today are:

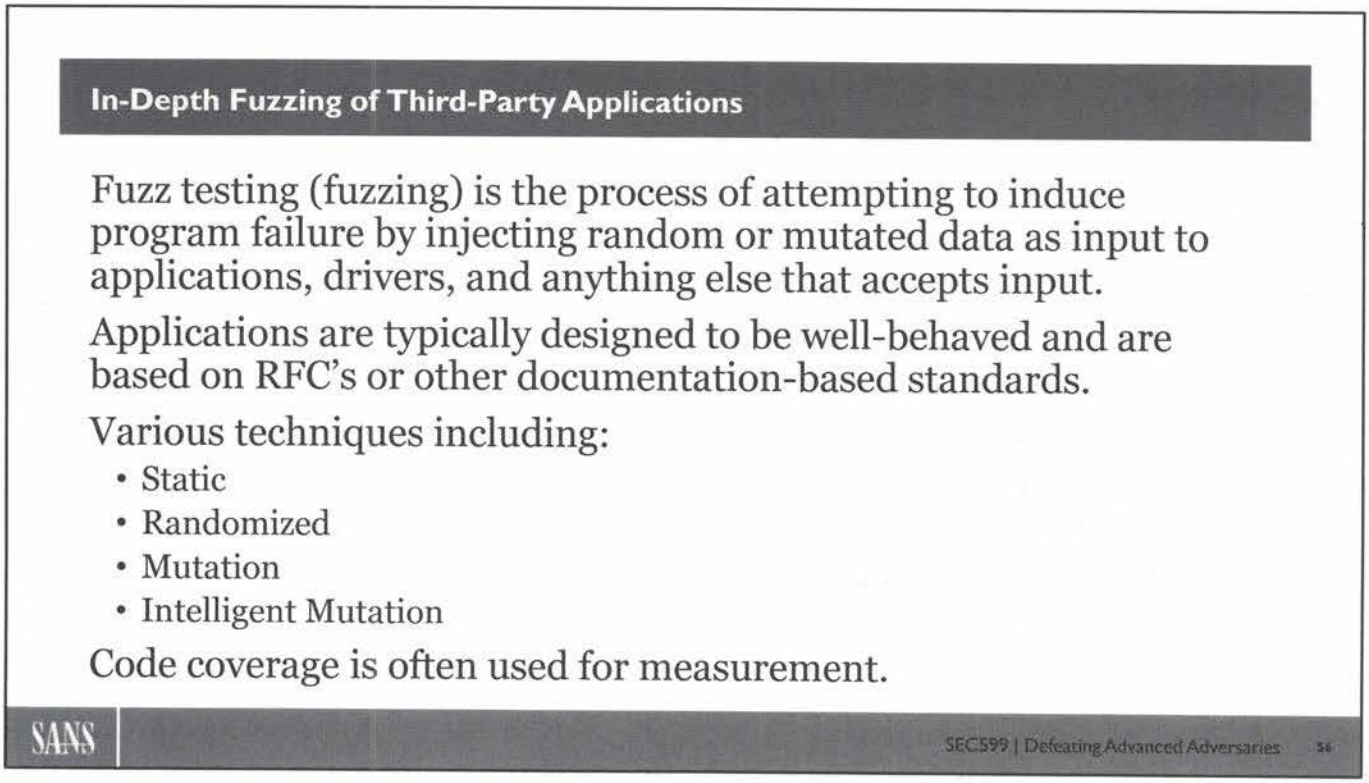
HP's Fortify Static Code Analyzer – HP acquired Fortify Software in 2010 which included the Static Code Analyzer. The tool supports more languages than any other solution available at the time of this writing. Languages include C, C++, Python, Ruby, Visual Basic, Java, PHP, and many others. More information at http://www8.hp.com/us/en/software-solutions/static-code-analysis-sast/

Veracode Static Analysis – The most common languages are supported, such as Java, C, C++, Objective-C, .NET, and others. More information at http://www.veracode.com/products/binary-static-analysis-sast

There are also a large number of open-source or free tools available for use. Two of the most popular:

CodeSearchDiggity – A tool included as part of the SearchDiggity project maintained by Bishop Fox that allows you to search through open source code for vulnerabilities and other related issues. More information at https://www.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/

VisualCodeGrepper – An actively maintained open source code scanning tool supporting multiple languages including Visual Basic, PHP, C++, Java, and a couple others. More information at https://sourceforge.net/projects/visualcodegrepp/




**In-Depth Fuzzing of Third-Party Applications**

Fuzz testing, also known as fuzzing, is the process of introducing random or malformed input to well-behaved network and file parsing applications, drivers, and anything else that accepts input. Googling for the definition of fuzzing will yield many different results. You have to imagine that the majority of applications are written based on a standard, such as those defined in a Request For Comment (RFC) document. Take the Session Initiation Protocol (SIP) defined under RFC 3261. SIP handles Voice Over Internet Protocol (VOIP) signaling, such as call setup and tear down. Many vendors, such as Cisco Systems and Avaya use such protocols. The protocol is standardized so that one vendor's product is compatible with another vendor's product. These RFC's tell you the rules, but do not tell you how to write the code. There are countless ways to write the code using many different languages. When writing a fuzzer you are attempting to test for various bug classes under many different conditions. You must think of, or have a tool do it for you, all of the ways a developer could have made a mistake and test to see if it exists. As you can imagine, this is a time-consuming process.

There are various fuzzing techniques that can be applied. Some of the most common testing techniques include static, randomized, mutation, and intelligent mutation.

Static – Manually develop test cases that check for a specific condition, such as the existence of a buffer overflow vulnerability in a field of an IPv6 header. Even for this one check, in this one field, you would want to test for multiple sizes for the input as you do not know how large of a buffer was created by the developer or what function they chose to use to copy data into memory. This is a time-consuming technique that requires a lot of familiarity with the protocol or file format being tested.

Randomized – This is a technique that requires little knowledge about the protocol or file format being tested. Fields are selected, and then random data is inserted into the field in an infinite loop until stopped.

Mutation – This is a technique that requires little knowledge about the protocol or file format being tested. Fields are selected, and then mutated data is inserted to test for conditions associated with various bug classes.

Intelligent Mutation – This technique requires deep familiarity with the protocol or file format being tested. Test cases are written to reach specific points in an application, and then selected fields are chosen for the introduction of mutated data. This is the most comprehensive technique if properly used. Tools can be used to measure the amount of code reached to ensure coverage is maximized.

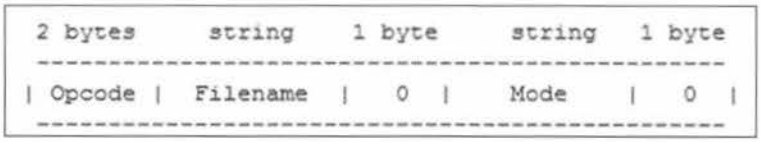
## Fuzzing Example: Trivial File Transfer Protocol (TFTP)

TFTP defined in RFC 1350:

- "TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP." (Sollins, K.)

This protocol will serve as our example when going through the various fuzzing types to help add context.

- We will use the following fields taken from the RFC which indicate the request type, filename, and mode:





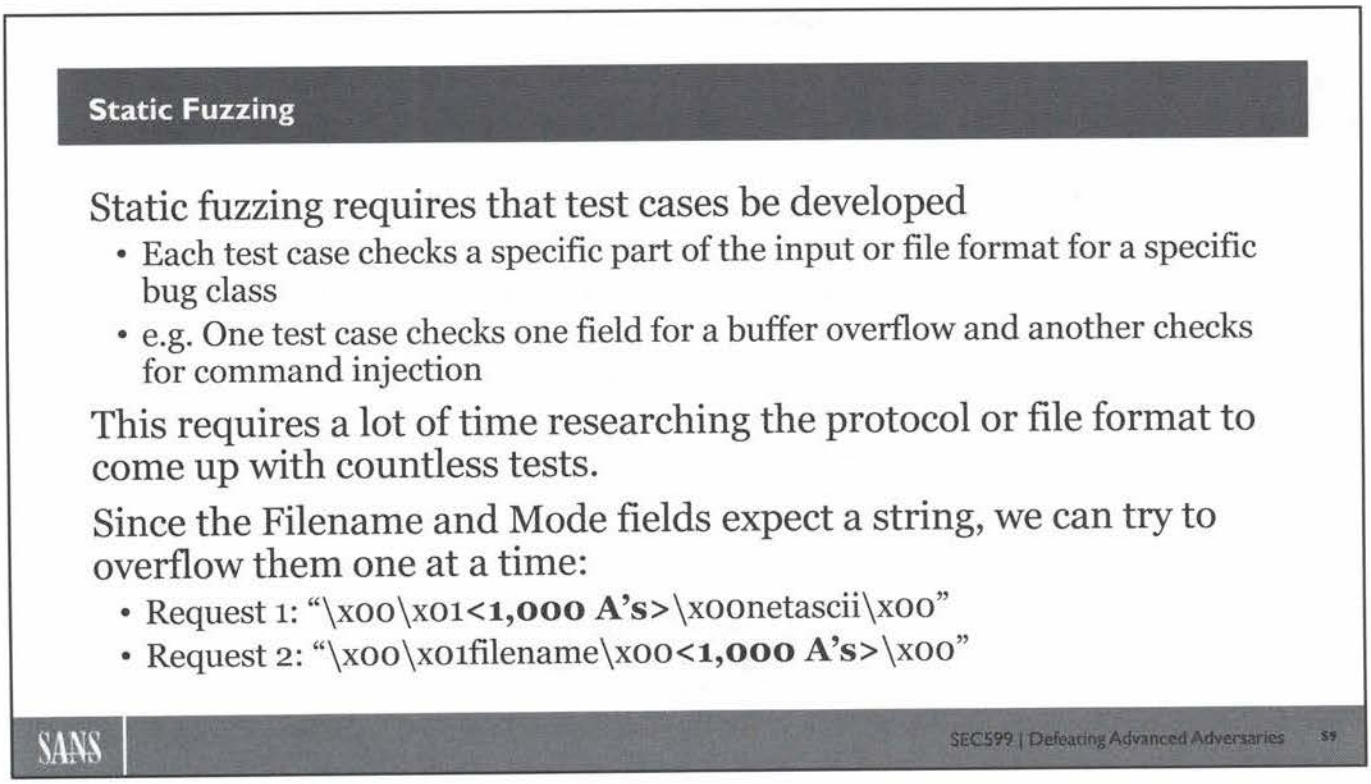

**Fuzzing Example: Trivial File Transfer Protocol (TFTP)**

The TFTP protocol is defined in RFC 1350. As stated in the RFC, "TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP." (Sollins, K.) This protocol was selected due to its simplicity. We will use it as an example for the various fuzzing techniques described shortly.

The image on the slide, taken from the RFC, shows the fields required when making a TFTP request. You would typically have a TFTP client that sends a request to a TFTP server. This request is most commonly a read or a write request. A read request means you would like to get something from the server and a write request means that you would like to put something onto the server. The 2-byte field for "Opcode" expects a "\x00\x01" for a read request and a "\x00\x02" for a write request. After this 2-byte field is the "Filename" field which expects a string. In other words, it wants you to include the name of the file to read or write. A question that must be asked is, "How does the server know when it's reached the end of the filename?" The next field is a 1-byte null or 0. When the server reads in the name of the file, it knows it's reached the end when it hits the null byte. The next field is the "Mode" field which is also to be a string. Options include "netascii," "binary," and "mail." Being that this field is also a string, it terminates with a null byte.

Citations and References:

Sollins, K. "The TFTP Protocol (Revision 2)." RFC 1350 The TFTP Protocol (Revision 2). https://tools.ietf.org/html/rfc1350 (accessed February 1, 2017).




**Static Fuzzing**

Static fuzzing requires the tester to spend a lot of time understanding the protocol to be fuzzed. Any documentation available, such as an RFC, would be inspected closely in order to begin creating test cases. Each test case would test for one condition. You have to remember that many languages can be used to write something such as a TFTP client or server. Each language comes with its own set of potential problems when poorly coded. Since we just saw that the Filename and Mode fields require a string, and keeps reading until reaching a null byte, that could be one field we'd want to check for the presence of a buffer overflow. On the slide, we have two examples:

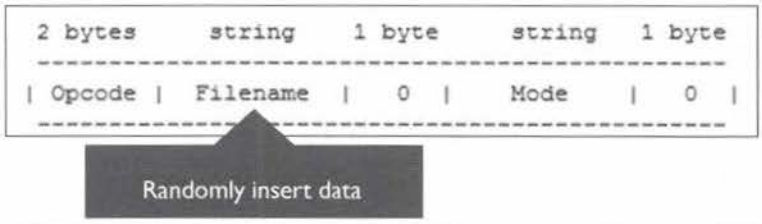
Request 1: "\x00\x01**<1,000 A's>**\x00netascii\x00"
Request 2: "\x00\x01filename\x00**<1,000 A's>**\x00"

Request 1 checks the Filename field to see if a string of 1,000 A's causes an overflow. Request 2 does the same, but for the Mode field. Just because a server doesn't crash with 1,000 A's doesn't mean you would move onto testing for another bug class. It might crash at 500 A's, but not 1,000, or any number of bytes below, between, or above. It all depends on how the code was written and any number of idiosyncrasies. Each program is different and is why we would need to come up with a large number of test cases that uses good samples to test for as many conditions as possible. This can result in thousands of test cases.

## Randomized Fuzzing

Randomized fuzzing is good from the perspective of not needing advanced knowledge of the protocol or file format being fuzzed.

- The tester simply selects a desired field to fuzz and the fuzzing application randomizes the input infinitely at that location
- Proper monitoring of the fuzzing session becomes critical using this technique
- If you crash the application but don't have monitoring you may never hit that condition again





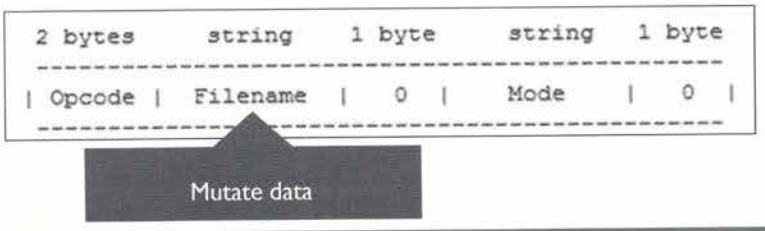

**Randomized Fuzzing**

Randomized fuzzing requires no upfront knowledge about the protocol or file format being fuzzed. Typically, a sample file or network packet data is acquired. This may be a valid PNG or JPG file, a captured SSH stream, or any number of options. A field is then selected to be fuzzed. The fuzzer then replays the network stream or produces a large number of files if fuzzing a file format, randomizing the data in the selected field. This would run infinitely until stopped. As you can see it is a simple concept that requires good monitoring to catch the crash. Take the example of fuzzing an SSH server. The fuzzer will continuously connect to the server sending random data in the selected field. If this causes a crash and the server dies how do you know what random data was sent to cause the crash? Since it is random, if you start it over you may never hit that test case again. It is therefore critical to record the data being sent to the server so that when it crashes, the most recent samples sent are available.

## Mutation Fuzzing

Mutation fuzzing also requires little knowledge of the protocol or file format being fuzzed.

- Instead of the data being random in the selected field, the data is mutated based on various bug classes
- It is finite in that there are only so many mutations to go through in each bug class being checked
- Once exhausted, the fuzzing session ends






**Mutation Fuzzing**

As with randomized fuzzing, mutation fuzzing requires little to no upfront knowledge about the protocol or file format being fuzzed. Where it differs is that instead of random data being inserted, mutations are generated based on various bug classes such as buffer overflows, command injection, and others. It is finite since there are only, so many mutations created per each bug class. This means that if you caused a crash, but missed the mutation that caused it, you could restart the fuzzer and likely hit the same condition.

## Intelligent Mutation Fuzzing

Intelligent mutation fuzzing can do all that of randomized and mutation fuzzing, but adds "intelligence".

- Often referred to as "protocol grammar," this fuzzing technique requires that the author scripts how the application communicates or handles files
- Take the PDF file format as an example, which is quite complex
- This fuzzing technique would require that you define the PDF file format in a script, and then choose which fields should receive mutation fuzzing

Regular mutation fuzzing typically starts with a sample data set and the field to fuzz, which will receive the mutations.

With intelligent mutation fuzzing, we script the file format upfront, giving us much more power and control.



**Intelligent Mutation Fuzzing**

Intelligent mutation fuzzing goes above and beyond the limitations of randomized and mutation fuzzers. It requires a lot of upfront time spent to understand the protocol or file format being fuzzed, similar to that of static fuzzing. With randomized and mutation based fuzzing a sample data set is typically used as a starting point. For example, if we wanted to fuzz the PDF file format we would start with a legitimate PDF document and select fields to mutate or randomize. With intelligent mutation fuzzing, we do not start with a sample data set. Instead, the file format or network protocol grammar is scripted into a source file. Some of the fields are left as static so they are not fuzzed, and others are selected for fuzzing. If an HTTP server must always receive a request type such as PUT, GET, POST, and HEAD, then we always want to lead with those values and don't want to fuzz them. Then, we can script out the rest of the interaction with the HTTP server, choosing desired fields for mutation. This allows us to get better code coverage.

## Introducing Peach

Peach Fuzzer Community Edition is an open source project that focuses on the individual hobbyist or researcher


- Available at http://www.peachfuzzer.com/resources/peachcommunity/
- Peach runs on Windows, Linux, and OS X
- Peach can fuzz files, network protocols & COM/DCOM
- A commercial version is also available

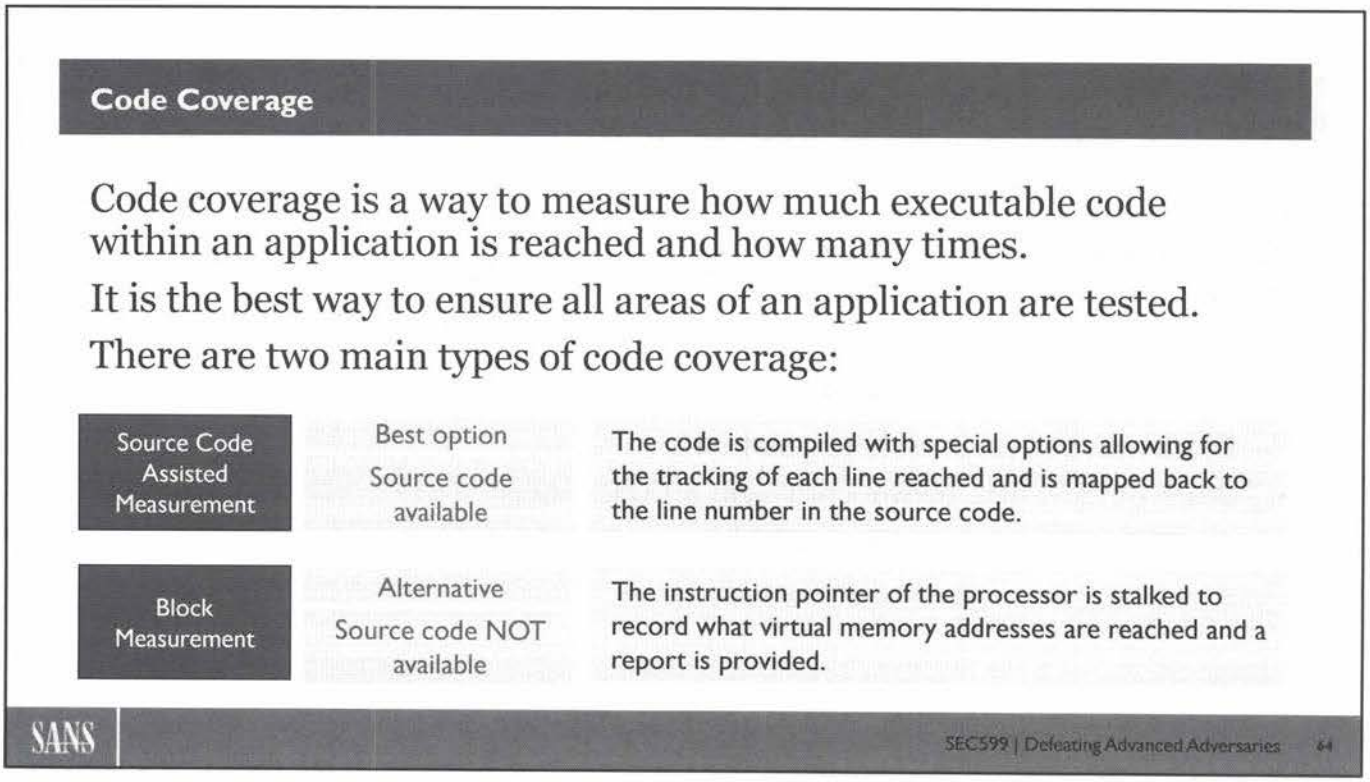


**Introducing Peach**

Peach Fuzzer Community Edition is an open source project that focuses on the individual hobbyist or researcher. As an open source project, changes largely consist of bug fixes with lengthy release cycles. Peach 3 is tested on Windows, Linux, and OS X. Peach 3 should also run on other platforms that Mono supports. Peach supports the following types of fuzzing:

- Stream-based fuzzing (files & sockets)
- Call-based fuzzing (COM/DCOM, RPC…)

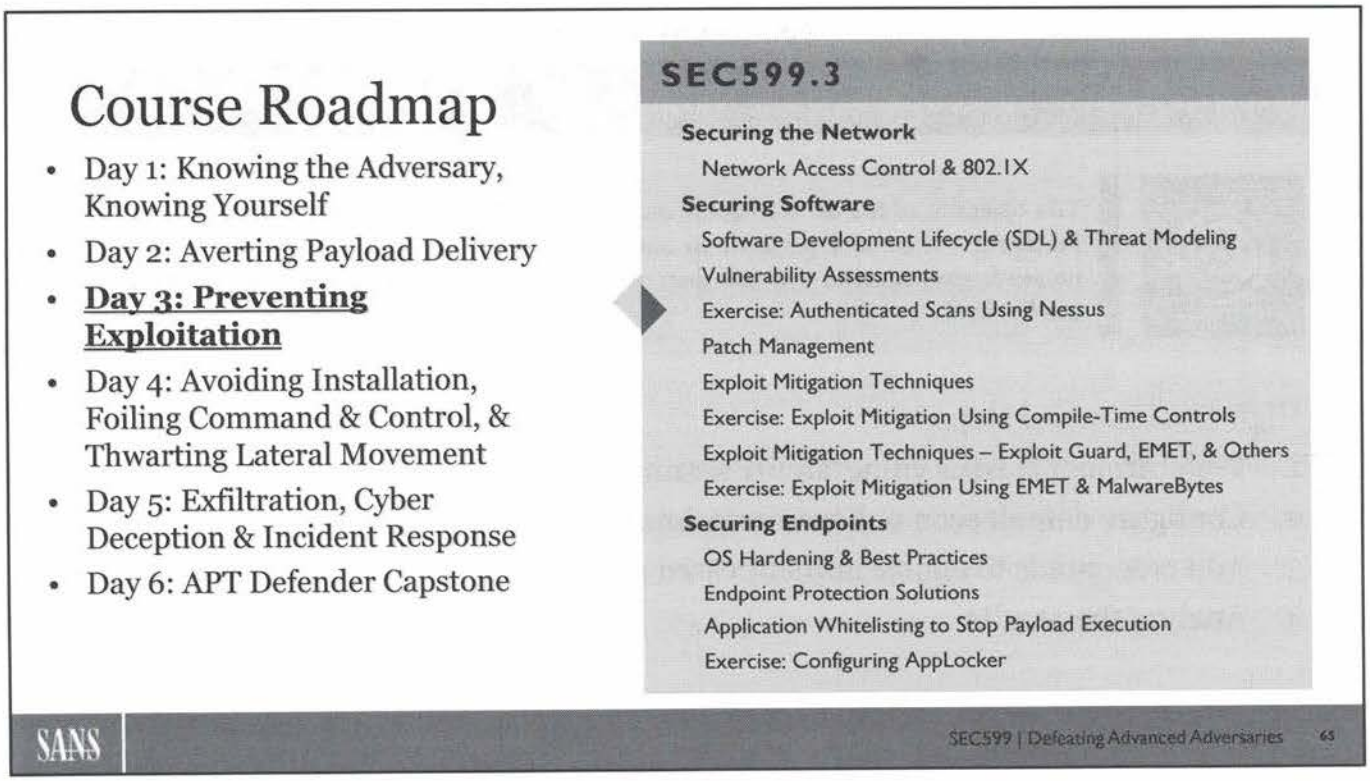
We can use Peach to assess software we don't own / know to see if we can find vulnerabilities by fuzzing. For more information, tutorials & downloads, please visit
http://www.peachfuzzer.com/resources/peachcommunity/



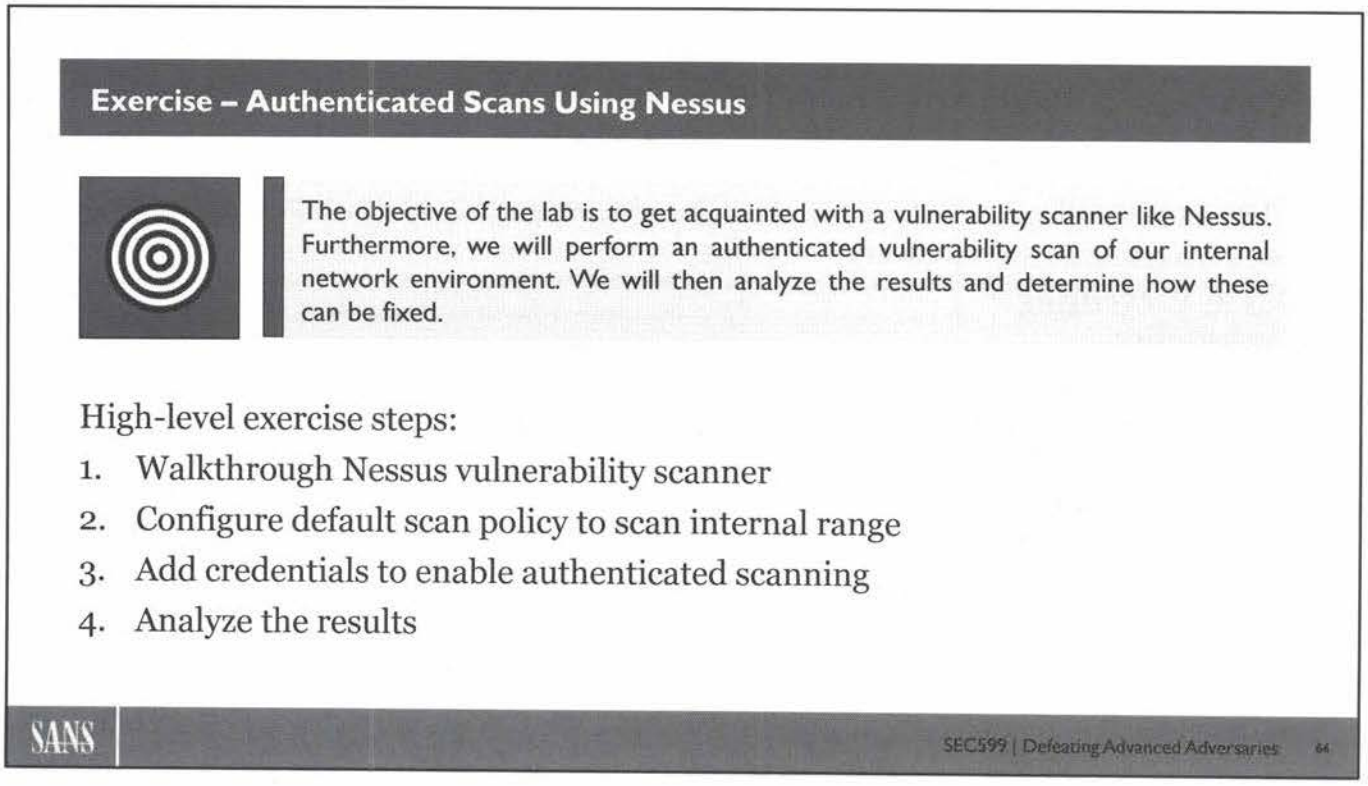

**Code Coverage**

Imagine if you were a law enforcement officer tasked with searching a large building for a suspect. It would likely be expected that you search each and every room. If you skip ¾ of the rooms as many are locked, then your coverage isn't very good. Now think of a large application such as Microsoft Word. There are many features and a lot of executable code. How do you know what lines of code you have reached, and which ones were not reached? You need a code coverage tool. The low-hanging fruit that is easy to reach in an application has likely been repeatedly tested and less likely to have a bug. The code that is harder to reach is more likely to have a bug.

There are two main forms of code coverage testing, source code assisted measurement and block measurement. Source code assisted measurement requires that you have the source code of the application being fuzzed, which is not always available. It also requires that you compile the source code with special debugging options to support the technique. There are various examples of source code assisted fuzzers and code coverage tools, such as the American Fuzzy Lop (AFL) by Michal Zalewski (lcamtuf) at Google. Source code line numbers are mapped to the compiled virtual addresses and then recorded at runtime during a fuzzing session. Tools such as the AFL also generate test cases (mutations) and automate the process. Block measurement can be used when source code is not available. A block is a grouping of code within a function that typically ends with a conditional branch, return, or other instruction. Imagine the statement, "If the value being checked is 0 go right and if not 0 go left." This branch would lead to another block of code within the function. In block measurement, each block's virtual memory address is recorded for later analysis.




This page intentionally left blank.




**Exercise – Authenticated Scans Using Nessus**
The objective of the lab is to get acquainted with a vulnerability scanner like Nessus. Furthermore, we will perform an authenticated vulnerability scan of our internal network environment. We will then analyze the results and determine how these can be fixed.

The high-level steps of the exercise are:

- Walkthrough Nessus vulnerability scanner
- Configure default scan policy to scan internal range
- Add credentials to enable authenticated scanning
- Analyze the results

For additional guidance & details on the lab, please refer to the LODS workbook.

## Exercise – Authenticated Scans Using Nessus - Conclusion

During this exercise, we used Nessus to perform vulnerability scanning against our internal network hosts:


Nessus is one of the most commonly used vulnerability scanning engines out there. Although it can never replace the expertise of a security researcher attempting to fuzz your custom applications for flaws, it can provide us with a good insight in misconfigurations & missing patches in our environment

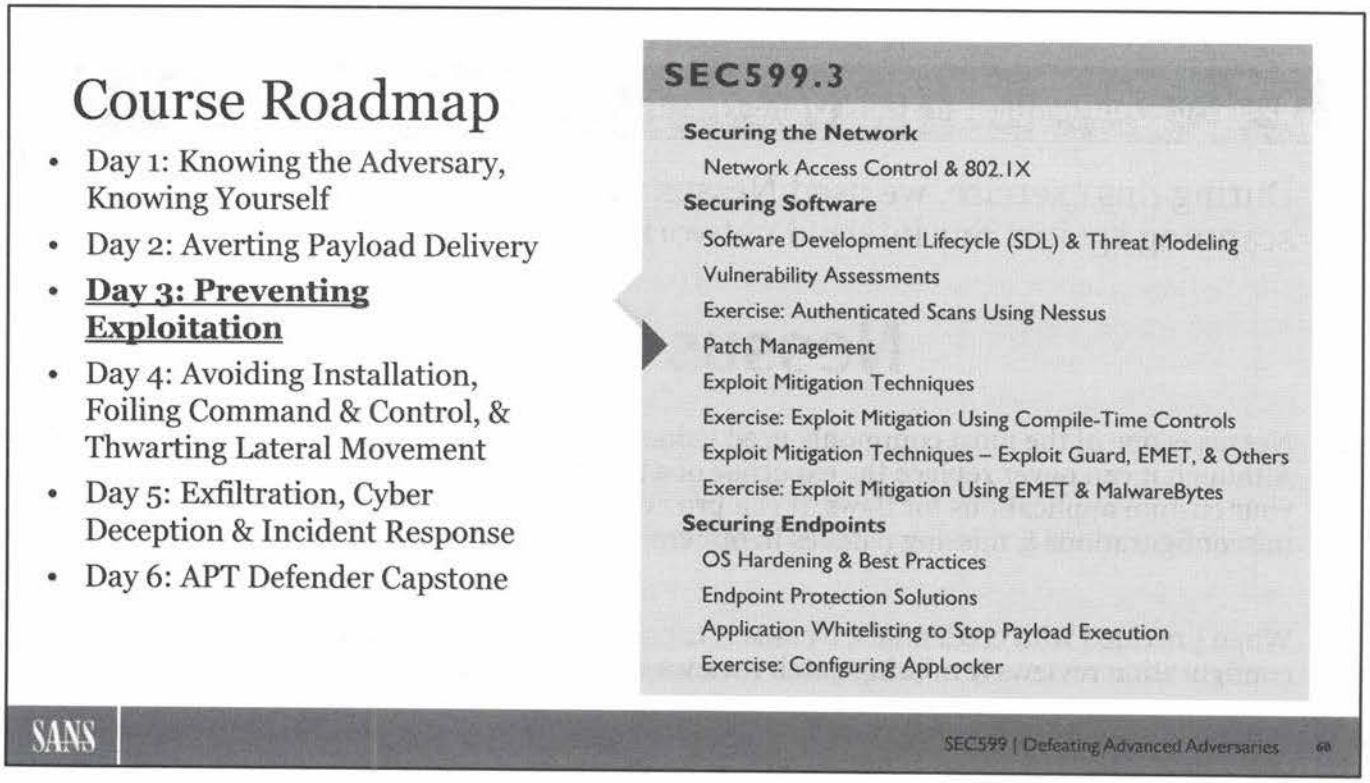
When provided with credentials, Nessus is capable of performing in-depth configuration reviews & missing patch reviews of a wide variety of Operating Systems



**Exercise – Authenticated Scans Using Nessus – Conclusion**

During this exercise, we used Nessus to perform vulnerability scanning against our internal network hosts:

- Nessus is one of the most commonly used vulnerability scanning engines out there. Although it can never replace the expertise of a security researcher attempting to fuzz your custom applications for flaws, it can provide us with a good insight into misconfigurations & missing patches in our environment.
- When provided with credentials, Nessus is capable of performing in-depth configuration reviews & missing patch reviews of a wide variety of Operating Systems.




This page intentionally left blank.

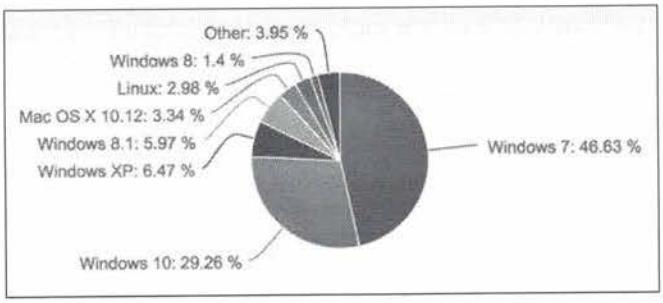
## OS Market Share

Windows 7 clearly dominant.

XP still at 6.47%
- Automated Teller Machines (ATMs)
- Embedded systems

Windows 10 quickly gaining traction.

Mac OS and Linux still a small number in comparison.




Taken on November 19th, 2017 from https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0

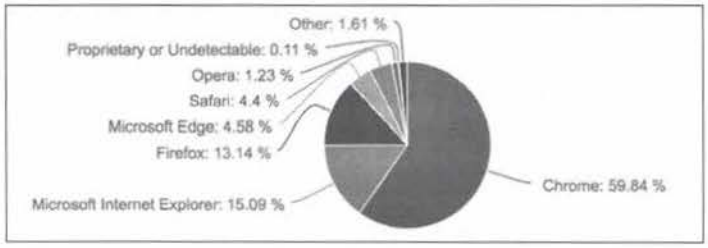


**OS Market Share**

The image on this slide gives you an idea as to the state of OS market share, clearly showing Microsoft Windows as the dominant OS. At the time this was pulled in November 2017, Windows 7 is by far the leader, with Windows 10 in second place. There are limitations in Windows 7 from a security perspective that cannot be fixed. An example is Control Flow Guard (CFG), a built-in security control we will discuss later this day. This was backported to Windows 8.1, but not Windows 7. There are also many improvements in the Kernel on Windows 10 that do not exist in Windows 7. It is critical to keep these older OSes up to date.

## Desktop Browser Market Share

- Chrome clearly the dominant browser.
- IE has lost market share over the past several years.
- Edge has not gained much traction.
- Firefox holding steady in third place.

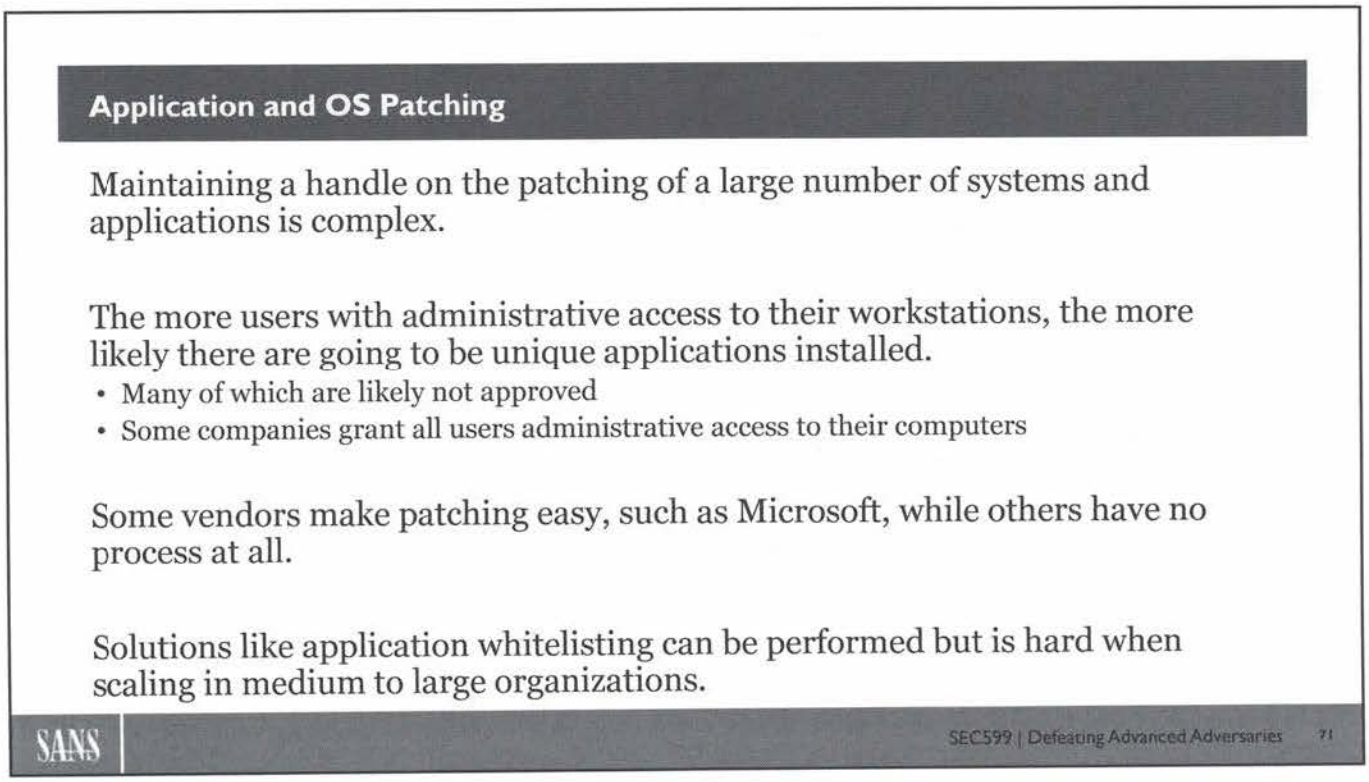



Taken on November 19th, 2017 from https://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomd=0



**Desktop Browser Market Share**

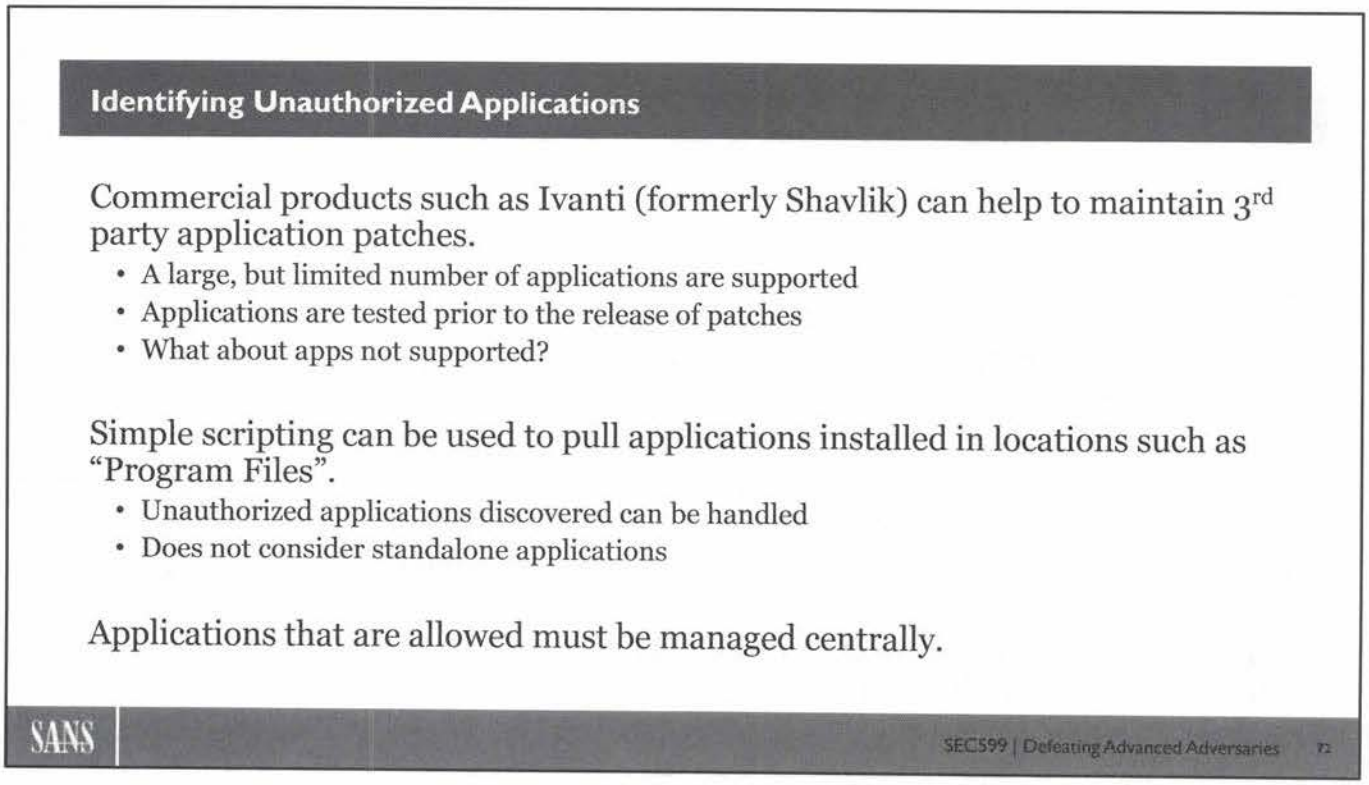
On this slide is the desktop browser market share as of November 2017. Note that these numbers change, so it is a good idea to use the provided link to see the updated percentages. For a very long time Internet Explorer was the dominant browser; however, Chrome has taken a large lead over the past couple of years. Microsoft Edge has struggled to gain traction. Browsers have been the source of many compromises over the years and as such, security has been greatly improved. Bug classes such as Use After Free (UAF) were one of the primary vectors in end-user compromises, but security improvements such as MemGC have greatly mitigated the technique.




**Application and OS Patching**

As we just saw it is no secret that the majority of companies use Microsoft Windows as their OS for employee workstations and laptops. Fortunately, Microsoft has a mature process for patch management with the well-known Patch Tuesday. Other OS vendors are not so clear as to when patches will be released, such as Apple. Some vendors, such as Oracle, have scheduled updates, but less frequently than Microsoft. Oracle releases updates each quarter. The mobile market makes things more complex, especially with some Android devices that are incapable of even being updated. Regardless, most organizations have a good handle on the patching of operating systems.

A different issue is the patch management of installed applications. Some very large organizations, who will not be named, give Administrative access to all employees. The reasoning for taking this approach is up to each company and their policies. A strong application whitelisting policy and enforcement can help mitigate the installation of unauthorized applications; however, whitelisting is difficult to manage. At organizations who do not grant all users administrative access, it is fairly typical for many users to be given approval to have this level of access. Take a network engineer as an example who is able to justify the need for Administrative access to perform job-related tasks. The justification may be valid; however, it often results in the installation of 3rd party applications that are not on the organization's approved list.




**Identifying Unauthorized Applications**

At one organization this author worked at, we wrote a script to pull the name and version of all applications installed on every system where the user had Administrative access. The result was a staggering 1,100+ applications. Some of these were the same application but different versions. When Googling for the words "<application name> vulnerability" there were countless hits. The point of this example is to demonstrate the complexity of trying to manage patches for that many applications not on the approved list. This also did not include standalone applications, but instead only applications installed in "Program Files" on Windows systems.

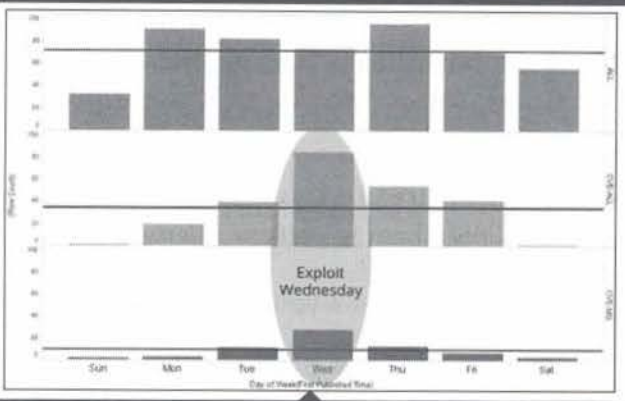
There are various commercial products available to aid in managing patches to 3rd party applications, such as Ivanti (formerly Shavlik). These solutions typically focus on the most common applications such as Flash, browsers, document readers, Oracle, etc.... Part of the service offered by these solutions includes the testing of the patches to ensure applications are not negatively affected. You would need a process for applications that are not supported.

More information:

http://www.ivanti.com/en-US/solutions/needs/patch-mgmt-for-systems-(os-and-third-party)

## Microsoft Patch Tuesday

- Microsoft releases patches on the **second Tuesday of each month**, for now...
- Effort to help **simplify the patching process**: random patch releases caused many users to miss patches.
- Patch delay of max. 30 days has security concerns: **emergency patches are released out-of-cycle.**
- Many **exploits** released in the days following a patch (="Exploit Wednesdays").




**Exploit Wednesdays**

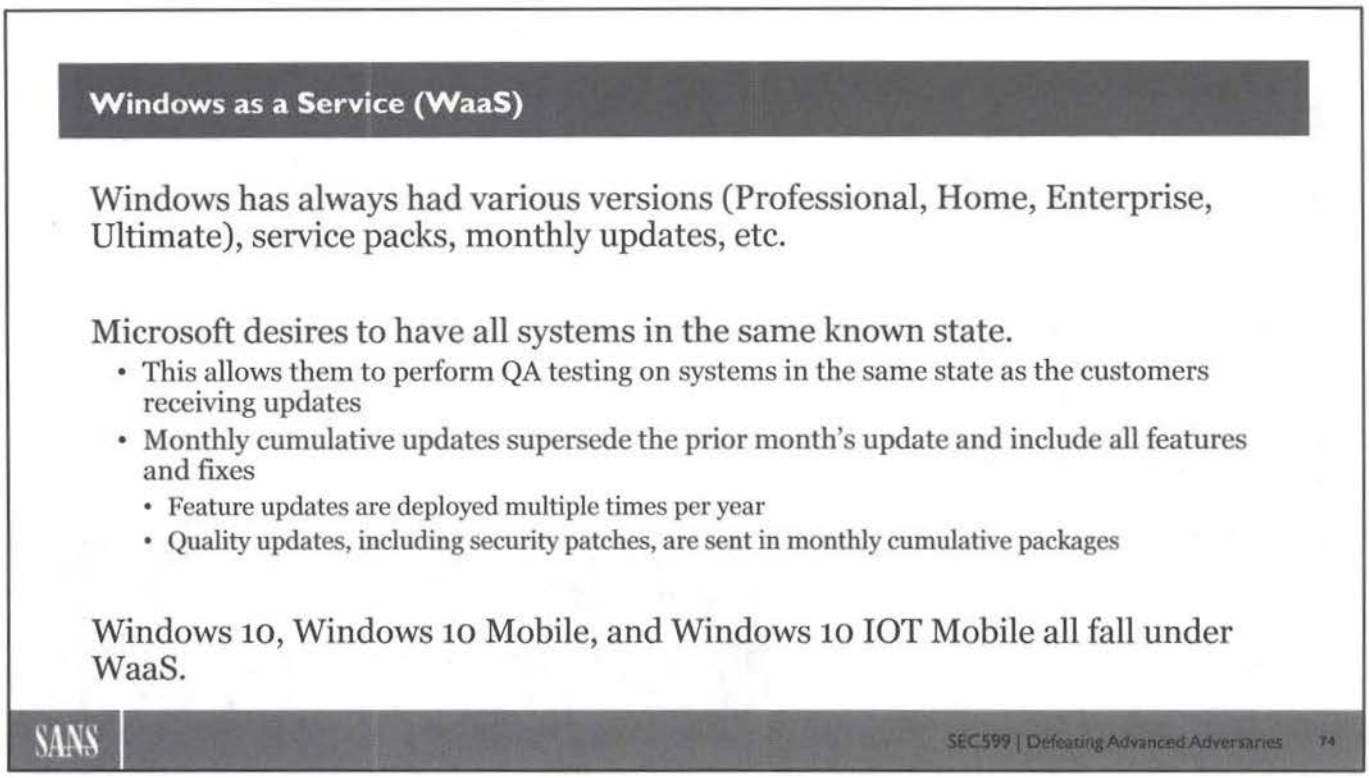
Analysis of a Russian hacker forum's traffic has shown evidence for exploit Wednesdays. While generic forum posts remain approximately constant throughout the week, there is a strong increase in traffic related to generic and Windows CVEs on Wednesdays.



**Microsoft Patch Tuesday**

In October 2003, Microsoft started its "Patch Tuesday" process. This came after many complaints from users and administrators who stated that it was difficult to keep up with patching their systems when it was unknown as to when patches would be released. The patches were released by Microsoft as they were approved. Users and administrators had to be constantly ready to handle the release of new patches. It is now well known that the second Tuesday of each month, Microsoft will release patches, both security-related and functionality or maintenance-related. The idea was that it would simplify the patching process for most organizations. Advanced alerts are sent out from Microsoft to inform and prepare users of the nature of each patch. Most organizations have adapted to the idea of "Patch Tuesday" and have a process in place to test patches, followed by deployment out to their systems. There are many services available to assist with patch deployment, from automatic updates on each Microsoft OS to Windows Server Update Service (WSUS) servers helping with large-scale patch management and deployment. Third-party applications are also available for patch management and deployment.

There are concerns about the waiting period in between patch releases from Microsoft. It is no secret that many exploit developers wait for patches to be released so that they can compare the patched version of a function or library to that of the unpatched version. Tools such as IDA Pro and BinDiff can quickly locate changes to the code. An experienced reverse engineer can locate the vulnerability within the unpatched code and write programs to reach the location within the affected program. This results in the release of cutting-edge exploits, which often prove lucrative to an attacker because many organizations do not quickly patch their systems. Exploits are sometimes released the following day after a patch is deployed by Microsoft, which caused the term "Exploit Wednesdays" to get adopted in the IT landscape. There is also the issue around attackers intentionally waiting until the day after patch Tuesday to release new unknown known exploits, knowing that it will likely not be patched for up to 30 more days. Microsoft does occasionally release out-of-band patches for critical updates; however, often systems are left unpatched for weeks. Workarounds are often provided, but this is only a temporary fix and is not always practical. Patch diffing is not only used by the bad guys. Those working for organizations often reverse engineer patches to determine the effect to the organization of patch application or to determine the impact of the vulnerability. Intrusion Detection System (IDS) signatures can also be developed from a thorough understanding of a vulnerability, as well as developing modules for vulnerability scanning and penetration testing frameworks.
Exploit Wednesday source:
Recorded Future
https://www.recordedfuture.com/hacker-forum-traffic/

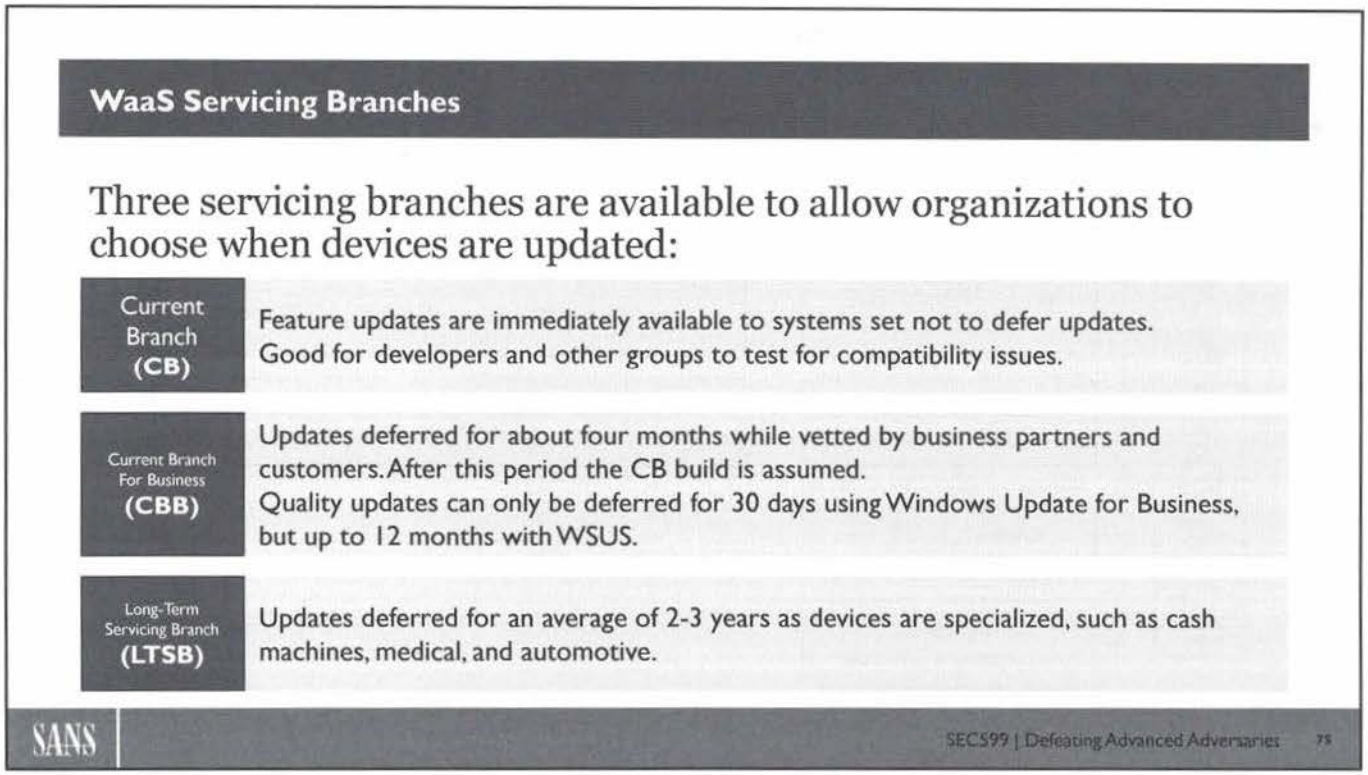



**Windows as a Service (WaaS)**
Prior to the release of Windows 10 Beta Microsoft started to release bits of information about the idea of Windows as a Service (WaaS). Little information was initially available, and many websites tried to define what it could mean. Even today, it is still complicated to digest with the introduction of servicing branches and various deferral options. Microsoft desires for all systems to be in the same state. This makes their life easier as the systems out in production around the world look more like the ones in their testing labs. Knowing the state and build of all systems out there should result in fewer compatibility problems. Each month a cumulative update is made available that supersedes the prior month's update. These cumulative patches include all updates for OS version. There are two types of updates: Feature and quality. They wish to do away with things like "Service Pack 2" and "Revision 3" and standardize on not more than two supported builds. The quality updates include security patches. Picking and choosing which updates to apply results in systems that are in many different states. Cumulative roll-ups help to ensure a system has all necessary patches to remain secure and support newer features.

Although cumulative updates are now pushed out for all systems, ever since October 2016, Windows 10, Windows 10 Mobile, and Windows 10 IOT Mobile are the only ones falling under the official WaaS practice.

References:
Halfin, Dani. "Overview of Windows as a service." TN Overview of Windows as a service. https://technet.microsoft.com/en-us/itpro/windows/manage/waas-overview (accessed January 29th, 2017).

More Information:
https://technet.microsoft.com/en-us/itpro/windows/manage/waas-update-windows-10




**WaaS Servicing Branches**

Three servicing branches are available from Microsoft to help with patch distribution. It actually gets quite complex when evaluating the various update server options such as Windows Update for Business, Windows Server Update Services (WSUS), and the System Center Configuration Manager. The references provided below and associated links at those locations are very useful in trying to understand how your Windows 10 organization can architect the right solution. The three branches are:

Current Branch (CB) – This branch makes features available as soon as they become available so that groups such as developers and QA can begin ensuring there are no compatibility problems, or those looking to take advantage of the new features can get started as soon as possible.

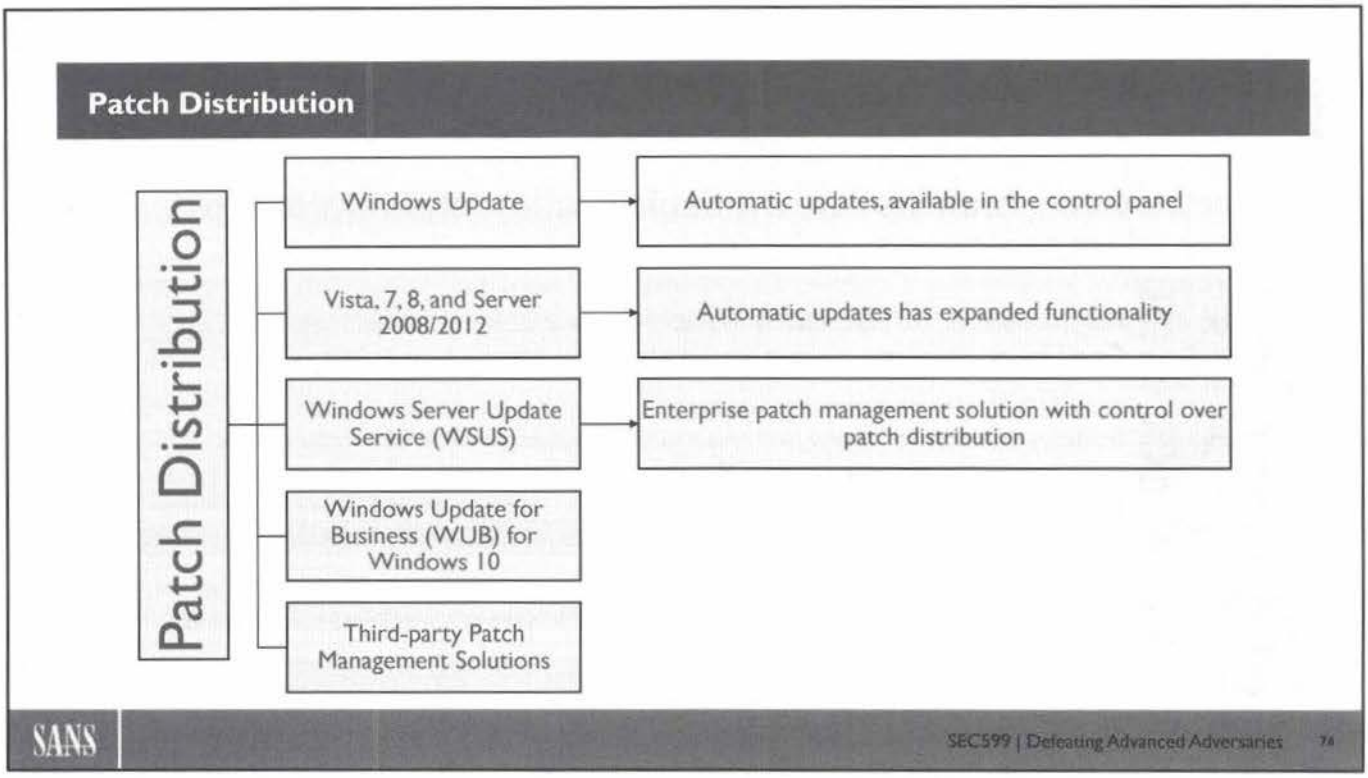
Current Branch for Business (CBB) – This branch is designed for wide-scale deployment to an enterprise. All of the features and such made available in the CB are moved to the CBB after a few month's vetting process involving customers and business partners. Quality updates can be deferred by different amounts depending on the Group Policy Option (GPO) settings pushed out and the patch distribution server option being used. This is likely going to continue to change and evolve as feedback is received by Microsoft.

Long-Term Servicing Branch (LTSB) – This branch is designed for specialized devices such as ATM/Cash Machines, medical devices, automotive devices and others. These are devices that have a specific focus or role and do not utilize the features made available by Microsoft.

References:

Halfin, Dani. "Overview of Windows as a service." TN Overview of Windows as a service. https://technet.microsoft.com/en-us/itpro/windows/manage/waas-overview (accessed January 29th, 2017).

Halfin, Dani. "Managing updates using Windows Update for Business." TN Managing updates using Windows Update for Business. https://technet.microsoft.com/en-us/itpro/windows/manage/waas-manage-updates-wufb (accessed January 29th, 2017).

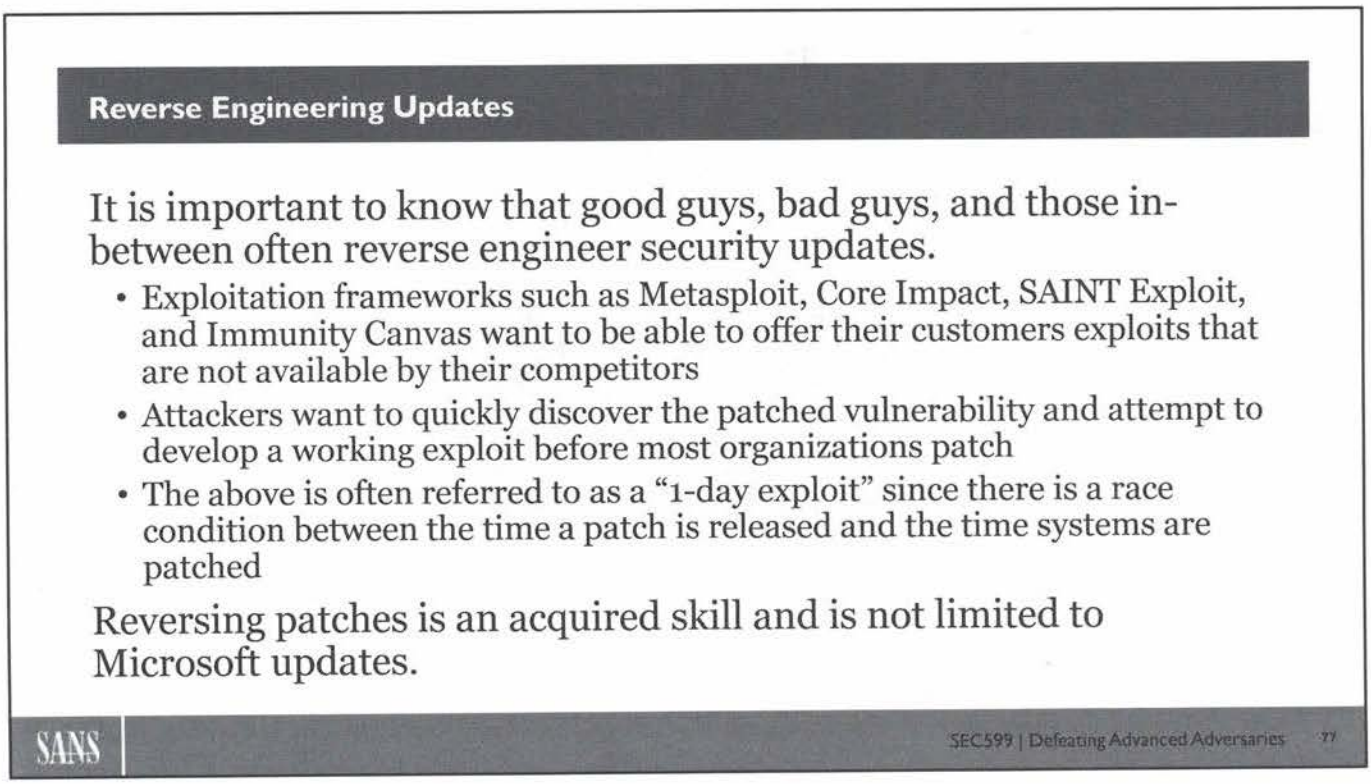



**Patch Distribution**

This slide serves as a simple high-level overview of the Microsoft patch distribution process. Many organizations do not permit end users to connect to Microsoft to obtain patches. Instead, a centralized enterprise patch management process controls patch distribution. The reasoning behind such a solution ranges from system consistency to security, to application stability. The ability for each user to connect at any time to the Microsoft update site and install desired patches renders the system builds to be highly inconsistent. Some patches have even been known to introduce new vulnerabilities. Other patches have been known to cause applications to break or behave differently than when the patch was not installed. All these issues make it desirable to control the distribution and installation of patches on end user systems and servers.

Automatic Updates has been installed by default on Windows systems since Windows ME, XP, and Windows 2000 Server. Automatic updates can be used to check for updates, check for updates and download them, and check for updates, download, and install them. Enterprise patch management often takes advantage of Windows Server Update Service (WSUS) servers to communicate directly with Microsoft update servers. Updates can be scheduled and sent directly to the WSUS servers over HTTP or HTTPS. Administrators then have the ability to first test the patches prior to deployment. Automatic updates on each end user system can be configured to communicate only with the enterprise WSUS servers. Administrators can select which patches they want pushed out and when. They also have the ability to set whether a patch can be deferred by the user and how soon a reboot is required if applicable. Windows Update for Business (WUB) is available starting with Windows 10. Update deferral is more limited, and Microsoft sees it as more of a constant stream of updates that should be installed as soon as possible. Check out SANS SEC505 "Securing Windows and PowerShell Automation" for more information on securely architecting Windows domains and building a patch management process.

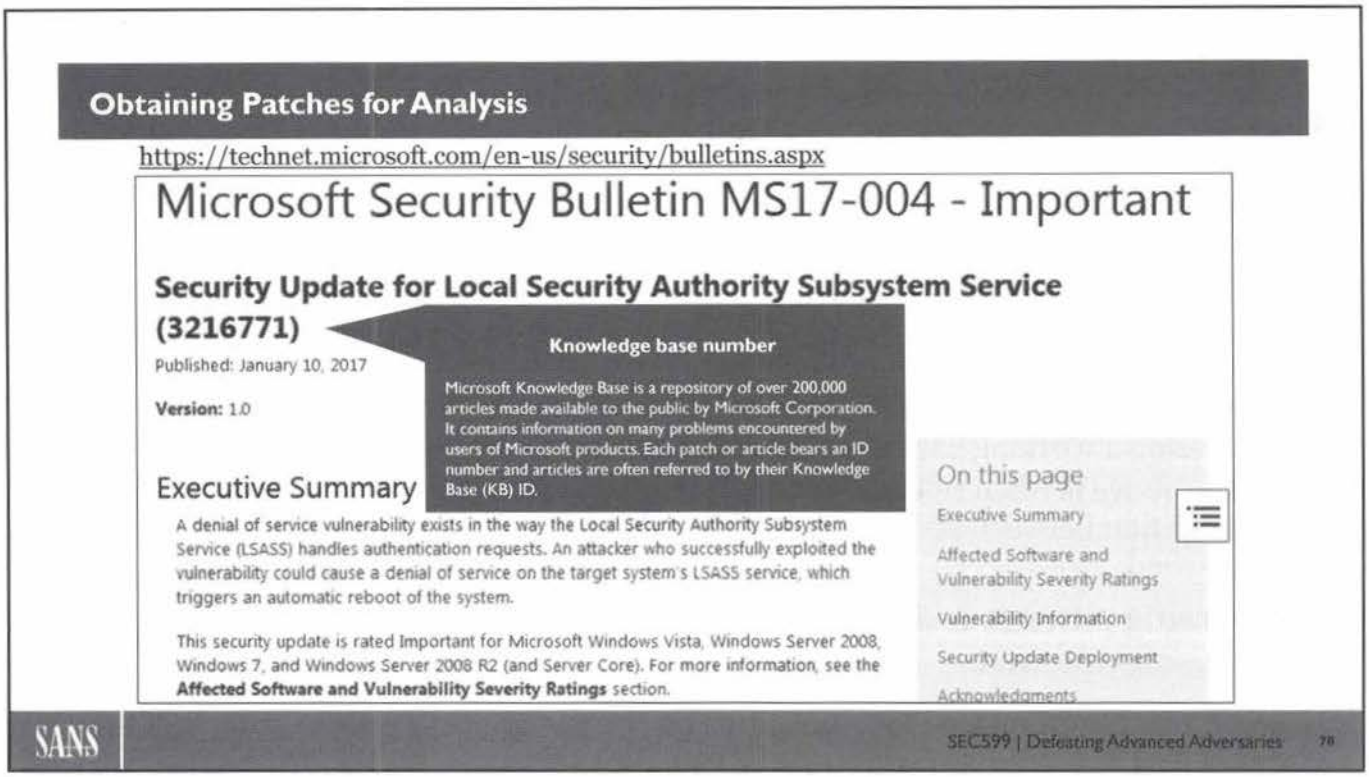
Third-party patch management solutions such as Patchlink and Lumension are available, often offering additional services and support for different operating systems.




**Reverse Engineering Updates**

If you think about a security update it should quickly become obvious that people might be interested in the contents of that update as it contains sensitive information to those with the right skillset. This is something that is performed by good guys, bad guys, and those in-between. Think about it from the perspective of a vendor who maintains an exploitation framework as a product, such as Immunity Canvas, Core Impact, Metasploit, or SAINT Exploit. Most disclosures are done privately and not found in the wild. This means that the vendor has been given the technical details and not the public. The vendor then creates a patch and distributes it to their customers. If you have someone with the expertise to take the patch and reverse engineer it to find the fix, that information can be used to potentially write a working exploit. Now your product has exploits available to privately disclosed vulnerabilities that are not possessed by your competitors.

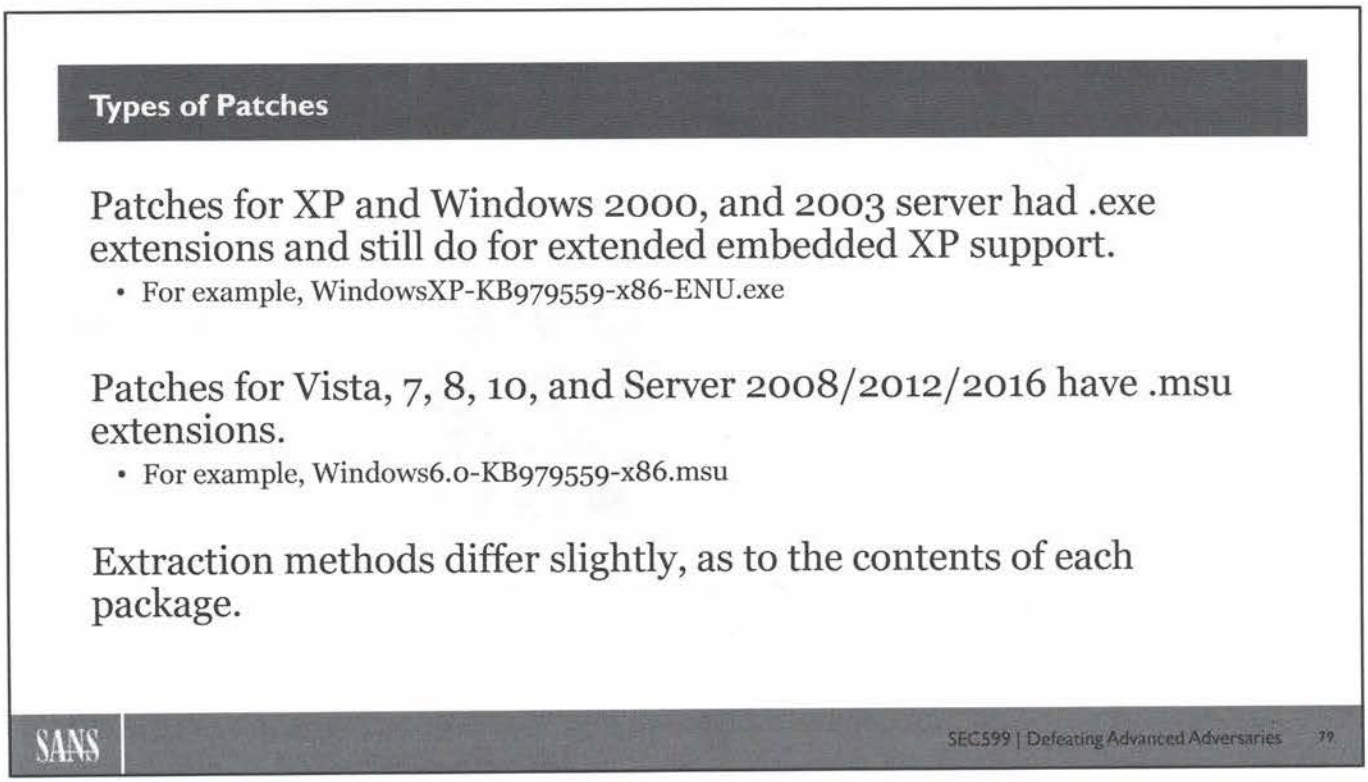
If Microsoft releases patches on the second Tuesday of the month, and then someone reverse engineers the patches and quickly gets an exploit working, that exploit would be valuable to penetration testers, attackers, and security vendors. This is often referred to as a 1-day exploit since those performing the reversing are attempting to quickly locate the fix and get a working exploit built before organizations patch. The more time that goes by the less valuable the exploit as more systems are patched.




**Obtaining Patches for Analysis**

Microsoft TechNet provides us with the ability to directly acquire patches. Available at https://technet.microsoft.com/en-us/security/bulletins.aspx, we can search for a specific update and download the appropriate patch for a given operating system. Patches are released in a couple different formats, depending on the OS level. The cumulative updates that started in October 2016 have made the process of identifying the individual patches more difficult. They used to be in a standalone format and easy to extract.

Microsoft Knowledge Base is a repository of over 200,000 articles made available to the public by Microsoft Corporation. It contains information on many problems encountered by users of Microsoft products. Each patch or article bears an ID number and articles are often referred to by their Knowledge Base (KB) ID.




**Types of Patches**

Most patches distributed by Microsoft have a .msu extension; however, legacy patches had a .exe extension. Patches for Windows XP, 2000 Server, and Server 2003 had the .exe extension, while Windows Server 2008/2012/2016, Windows Vista, and Windows 7/8/10 have the .msu extension. For example, a patch for a Windows XP system would look like

WindowsXP-KB979559-x86-ENU.exe

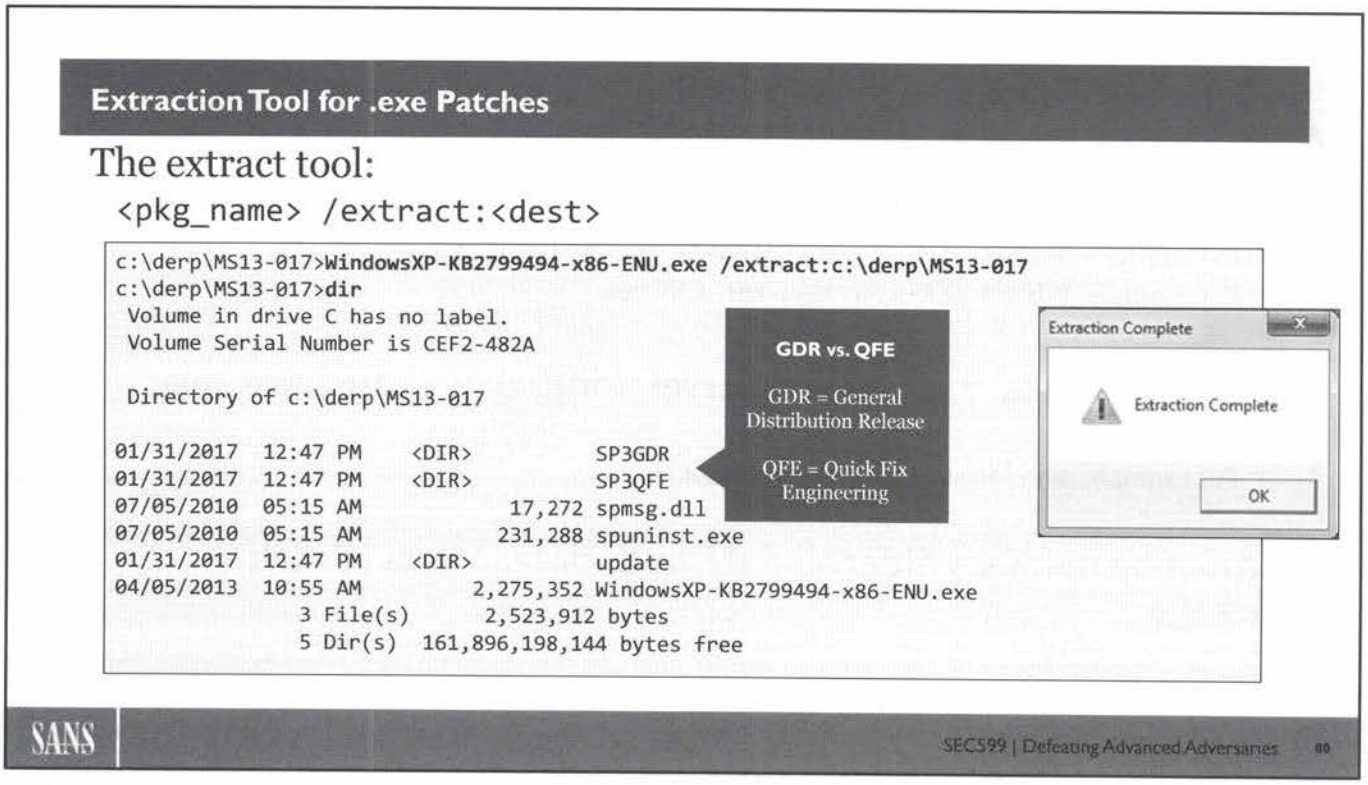
The same patch on Server 2008 would look like:

Windows6.0-KB979559-x86.msu

You may be wondering why it's worth mentioning XP. Good question. Patches for XP embedded are still available until April 9th, 2019. There are people out there who acquire the embedded version patches, use a registry hack, so that they can stay on XP. See this website as an example: http://www.ghacks.net/2014/05/24/get-security-updates-windows-xp-april-2019/

From the perspective of reverse engineering patches XP is also still of interest. Many vulnerabilities affect many or all versions of Windows. You would have to imagine that the code on XP is likely less complex than Windows 10 and therefore, reversing the same vulnerability on XP could save time.

Contents within the patch files differ depending on the OS, as do the tools to extract them manually. The .exe patch files tend to be much simpler to get to the wanted files, whereas the .msu patch files may require additional examination, especially with cumulative updates.



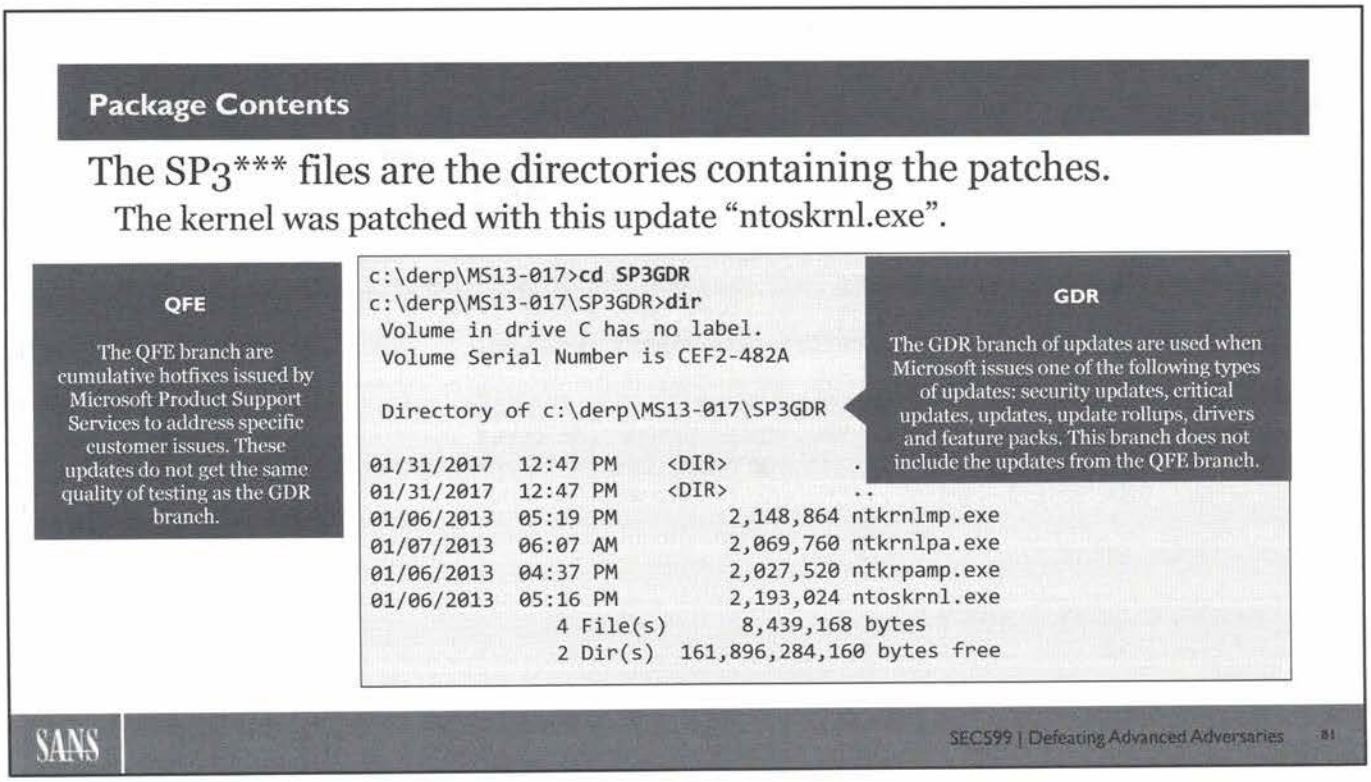

**Extraction Tool for .exe Patches**

The extract tool can be used via the command line to extract patches with the .exe extension. Simply type in the name of the patch file containing the .exe extension, followed by /extract:<dest>. For example:

C:\derp/MS13-017> **WindowsXP-KB2799494-x86-ENU.exe /extract:c:\derp\MS13-017**

If successful, you get the pop-up box on the screen stating that extraction was successfully completed. Proceed to review the contents of the package.

You may have noticed that there are two folders: one with GDR in the title and the other with QFE. GDR stands for General Distribution Release and QFE stands for Quick Fix Engineering.

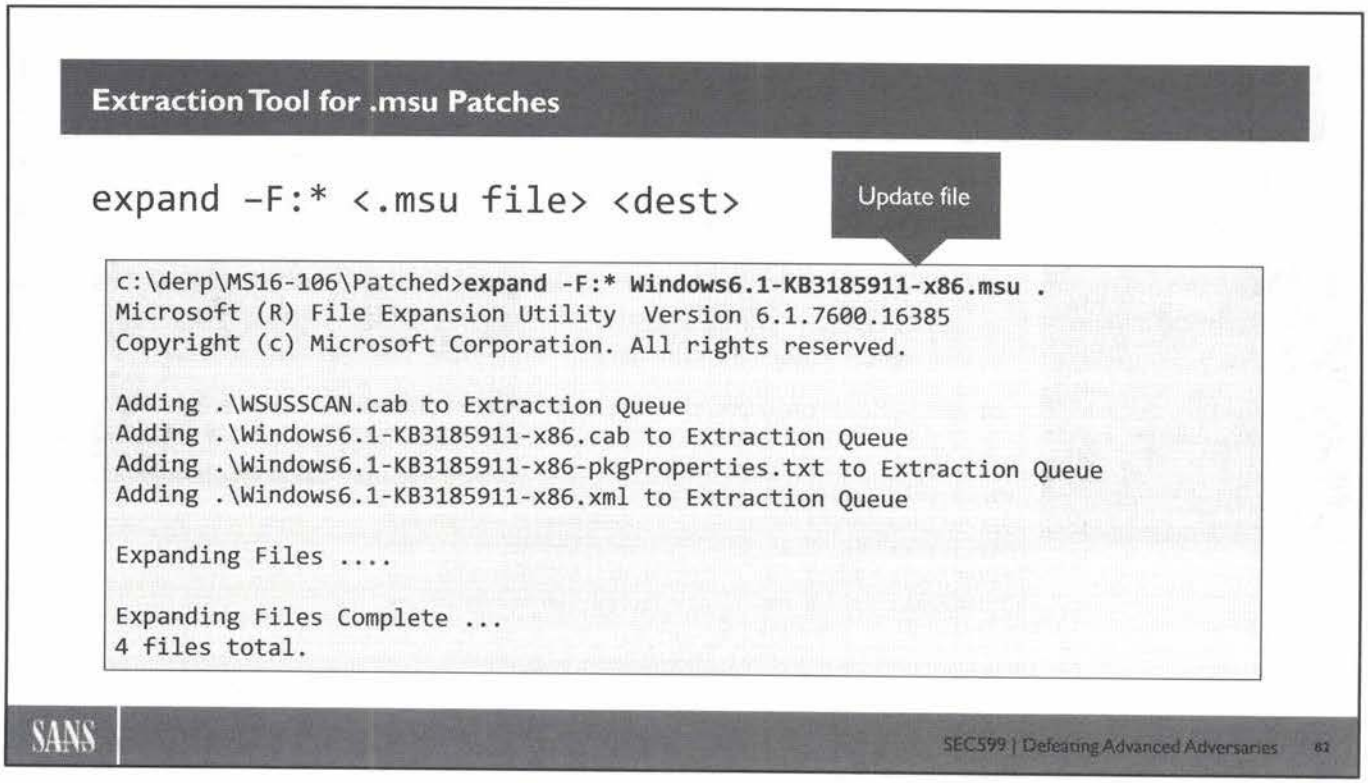



**Package Contents**
The package contents of this update are shown in the screenshot. As you saw on the last slide, there are two directories listed for XP SP3 called SP3GDR and SP3QFE. The contents of the directory SP3GDR contains multiple files, such as "ntoskrnl.exe." This is actually the name of the Windows Kernel and therefore the Kernel was patched in this fix. Command switches were used to limit the output to fit the image onto the slide.

"The GDR branch of updates are used when Microsoft issues one of the following types of updates: security updates, critical updates, updates, update rollups, drivers, and feature packs. This branch does not include the updates from the QFE branch.

The QFE branch is cumulative hotfixes issued by Microsoft Product Support Services to address specific customer issues. These updates do not get the same quality of testing as the GDR branch."

Citation:
Jphillips59. "QFE vs. GDR." QFE vs. GDR Microsoft (Windows) Support – Neowin Forums. https://www.neowin.net/forum/topic/332694-qfe-vs-gdr/ (accessed January 31st, 2017).

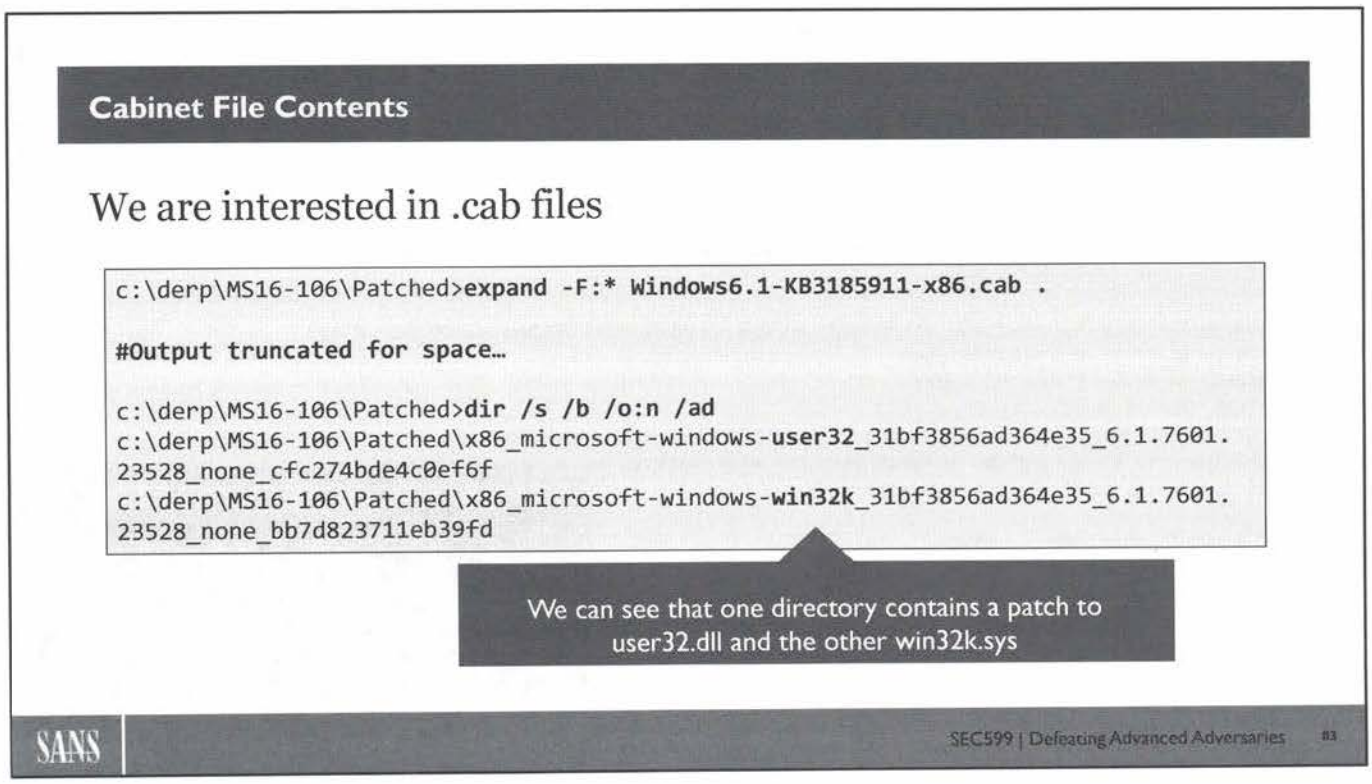



**Extraction Tool for .msu Patches**

For Windows Vista, 7, 8, 10 and Server 2008/2012/2016, the expanded tool can unpack packages with the .msu extension. As shown on the slide, the file Windows6.1-KB3185911-x86.msu is expanded with the following command:

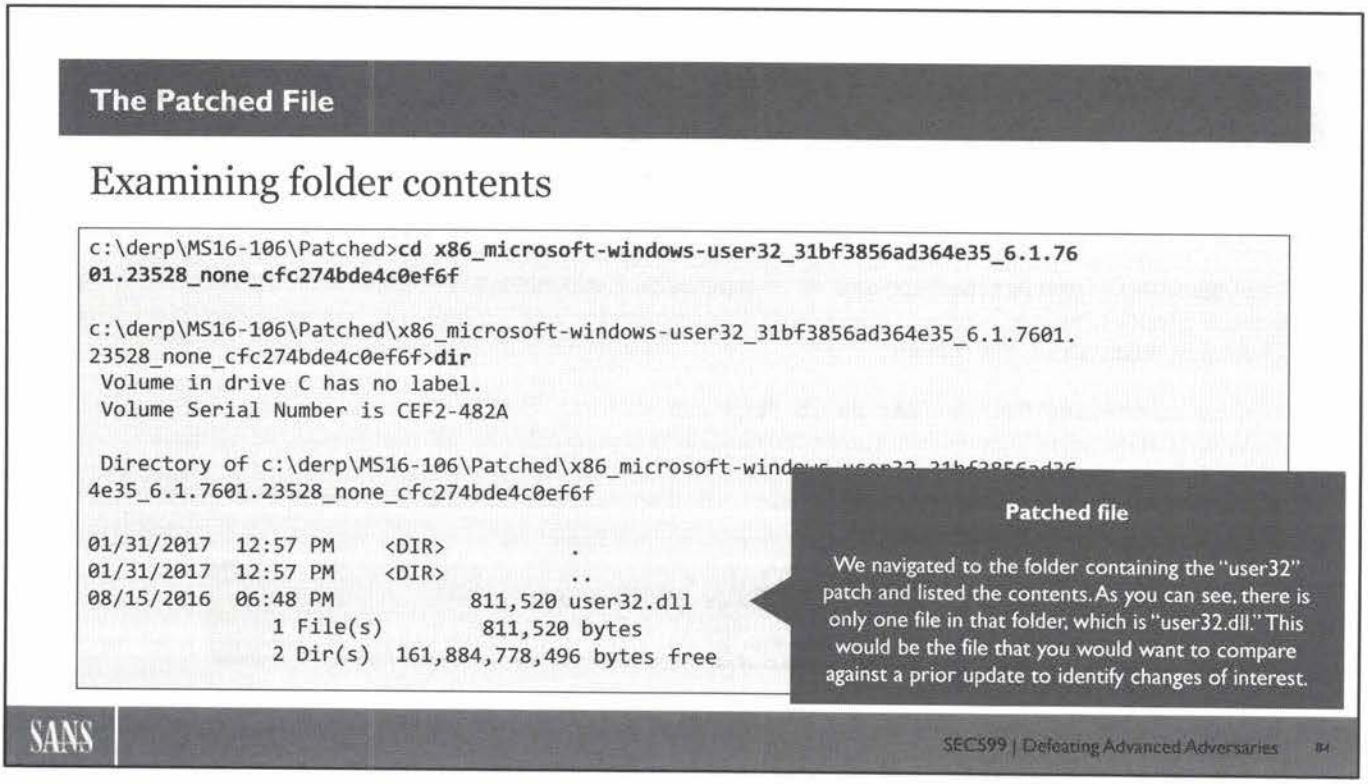
expand –F:* Windows6.1-KB3185911-x86.msu .

Four files are unpacked and can be seen.




**Cabinet File Contents**

As seen on the prior slide, several files were extracted from the .msu file. We must now use the same method to extract the .cab file. A lot of output is displayed on the screen when extracting the .cab file and as such, it was truncated from the output on the slide for spacing purposes. A customized "dir" command is then issued to limit output to directories only. You can see there are two folders, one containing a reference to the name "user32" and the other "win32k."




**The Patched File**

We have now simply navigated to the folder containing the "user32" patch and listed the contents. As you can see, there is only one file in that folder, which is "user32.dll." This would be the file that you would want to compare against a prior update to identify changes of interest. More on this shortly.

## Extracting Cumulative Updates

As mentioned previously, patches are now cumulative and contain all updates for the OS version.

- This makes for very large update files that contain hundreds of files
- Mapping an extracted file to the right Knowledge Base (KB) number is difficult

Greg Lineras (@Laughing_Mantis) wrote some PowerShell scripts to help with this problem.

- The concept is quite simple: using the modified data on the updates to identify files that have changed within the last 30 days
- They are then placed into unique directories and cleanup is performed
- You still need to determine which file correlates to which advisory, but the process is much easier



**Extracting Cumulative Updates**

As previously mentioned, the cumulative updates are very large and contain all patches for the OS version. When extracting them there are hundreds to thousands of files. This makes it very challenging to sort through them to find the desired file and map it correctly to the Knowledge Base (KB) number. Greg Lineras, known as @Laughing_Mantis on Twitter, created a couple of PowerShell scripts to help with this issue (you can find an example here: https://pastebin.com/0mYXJGC5). The idea is quite simple, extract everything, delete all the junk we do not care about, and sort the files over 30 days old into an "old" directory. This allows you to focus on the files that have a modified data within the past 30 days. You still need to map the files to the correct KB number, but now you are only looking at ten or so folders as opposed to a very large amount.

## Obtaining a Cumulative Update for Windows 10

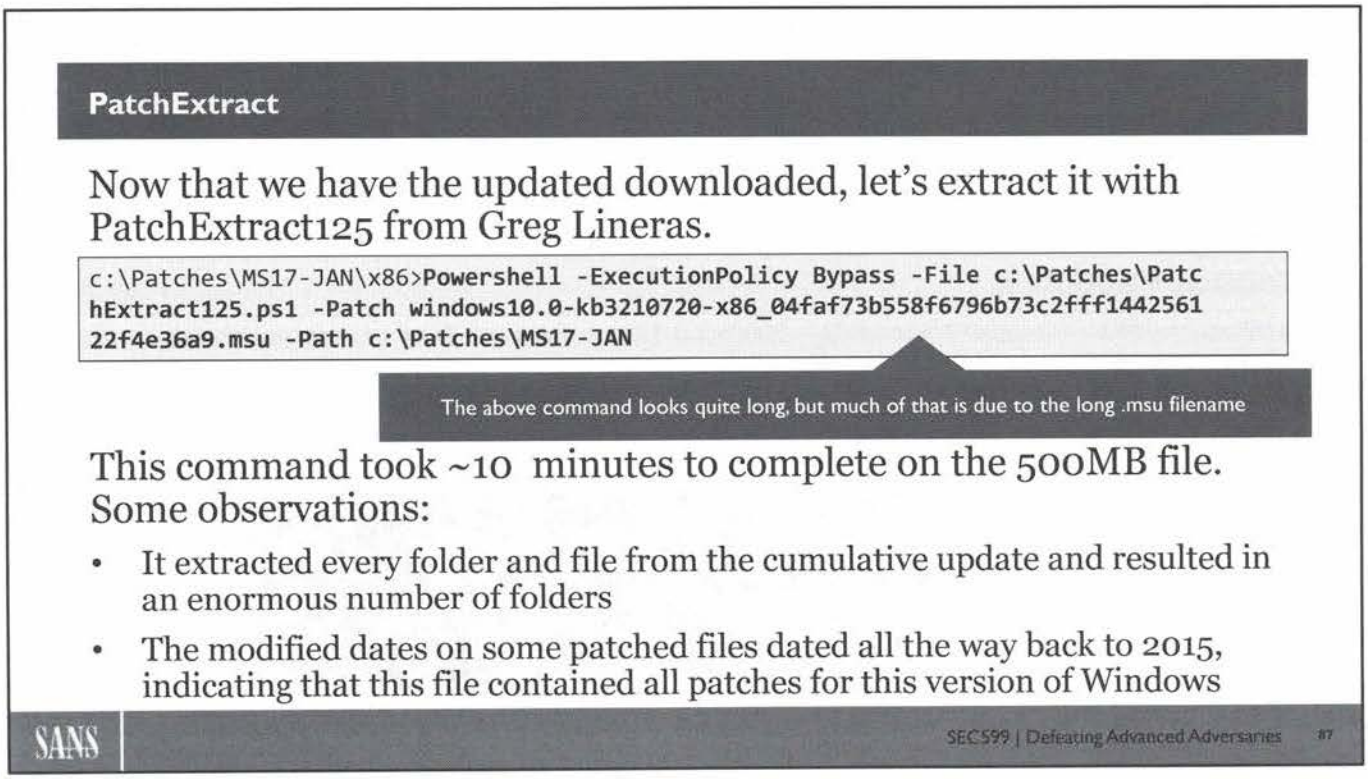
The following screenshot shows the cumulative update file for January 2017:


The cumulative updates result in some very large files.



**Obtaining a Cumulative Update for Windows 10**

This slide simply shows a screenshot of the January 2017 cumulative update for Windows 10. As you can see the x86 file is 490MB and the x86 file is around 1GB. Changes in the way Microsoft is managing incremental updates to reduce the file size may allow for this to be smaller for patching: http://www.theverge.com/2016/11/3/13511012/microsoft-windows-10-unified-update-platform-features



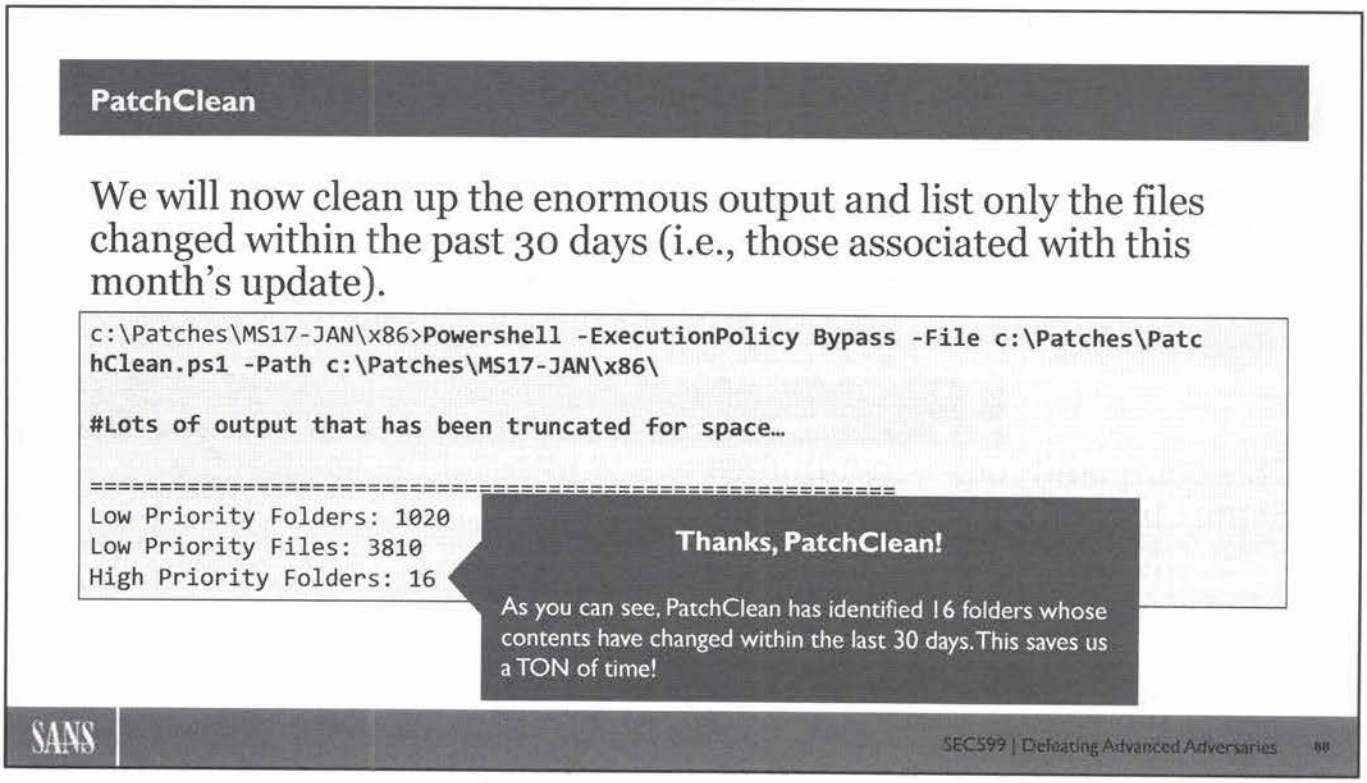

**PatchExtract**
With the update downloaded, let's extract it with the PatchExtract tool from Greg Lineras.

C:\Patches\MS17-JAN\x86>**Powershell -ExecutionPolicy Bypass -File c:\Patches\PatchExtract125.ps1 -Patch windows10.0-kb3210720-x86_04faf73b558f6796b73c2fff144256122f4e36a9.msu -Path c:\Patches\MS17-JAN**

The command is rather long due to the .msu filename; however, we're simply telling it what script to execute "PatchExtract125.ps1," then the name of the .msu file with the "-Patch" switch, and then the path where to put the extracted files with "-Path."

Depending on the size of the .msu file (500MB in this case) it can take quite a while to extract all of the files. It took ~10 minutes for this file. The result is excluded from the slide as it is quite a lot of output, as well as over a thousand files and folders. When looking at a couple of sample files the dates were all the way back into 2015, showing that the .msu file contains all patches for this version of Windows.

PatchExtract can be found at: http://pastebin.com/u/Laughing_Mantis



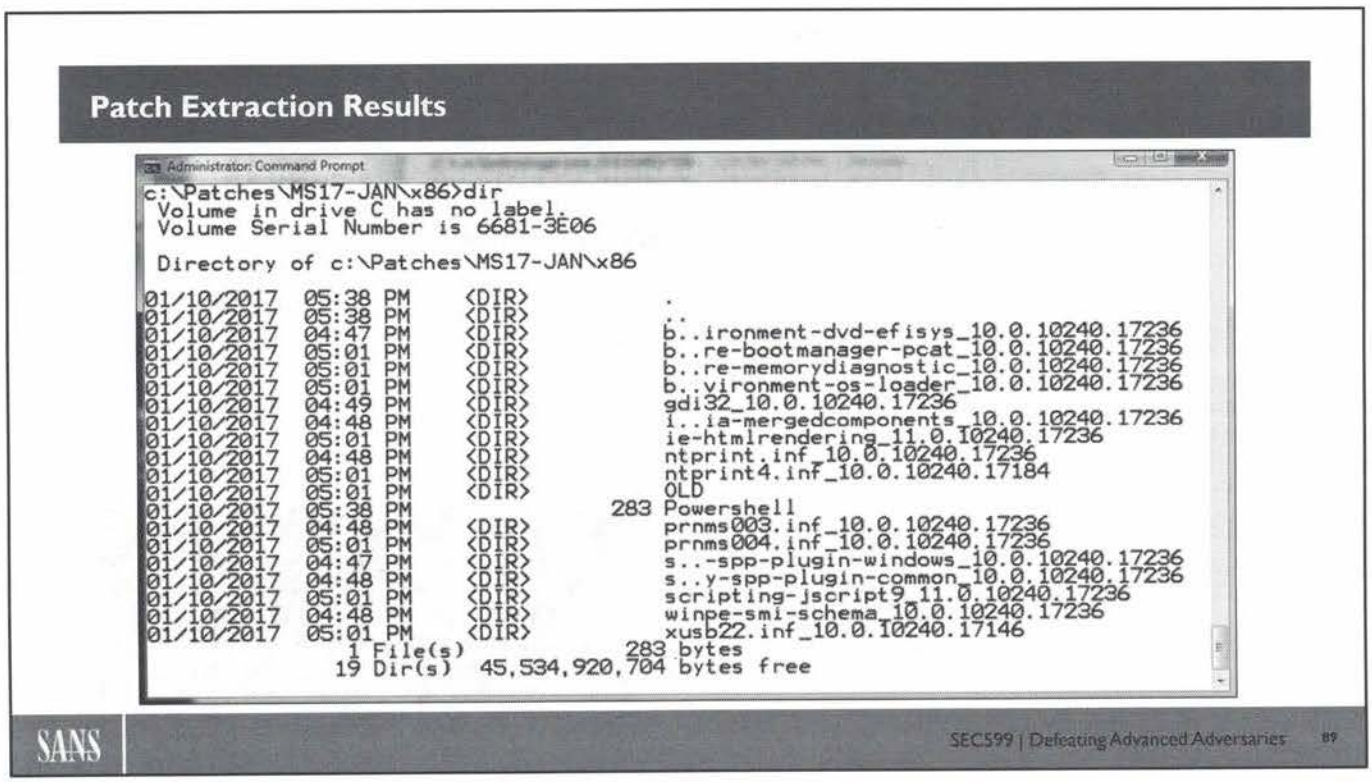

**PatchClean**

With all of the files and folders from the cumulative update extracted, we want to know which ones are associated with this month's update. The PatchClean script will go through and put every folder that contains a folder with a "Date Modified" time of >30 days into a folder called "Old." It will leave only folders with files in them that have a "Date Modified" time within the last 30 days. We run PatchExtract with:

c:\Patches\MS17-JAN\x86>**Powershell -ExecutionPolicy Bypass -File c:\Patches\PatchClean.ps1 -Path c:\Patches\MS17-JAN\x86**

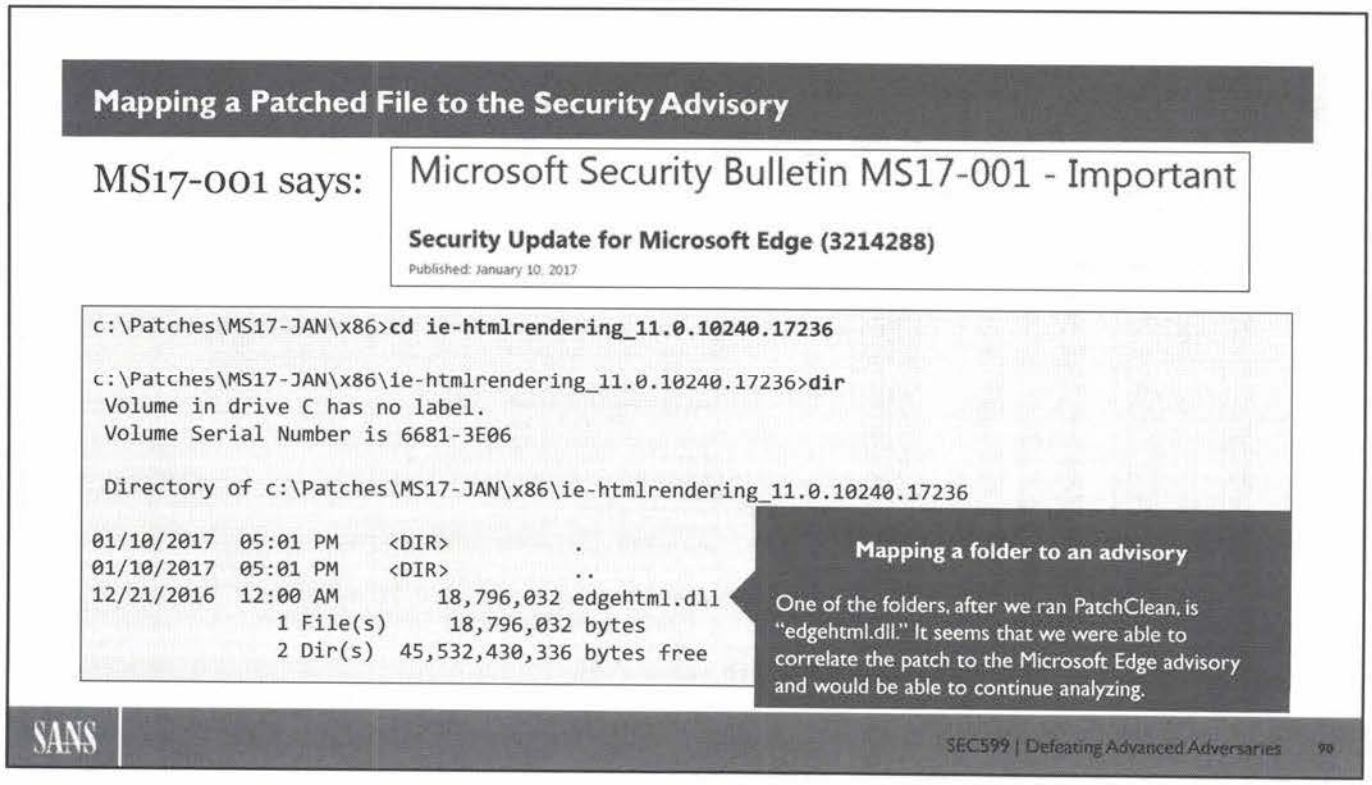
The result, as shown on the slide, is 16 high priority folders. That is much less than the >1,000 folders and >3,800 files extracted from the cumulative update.

PatchClean is also available at: http://pastebin.com/u/Laughing_Mantis



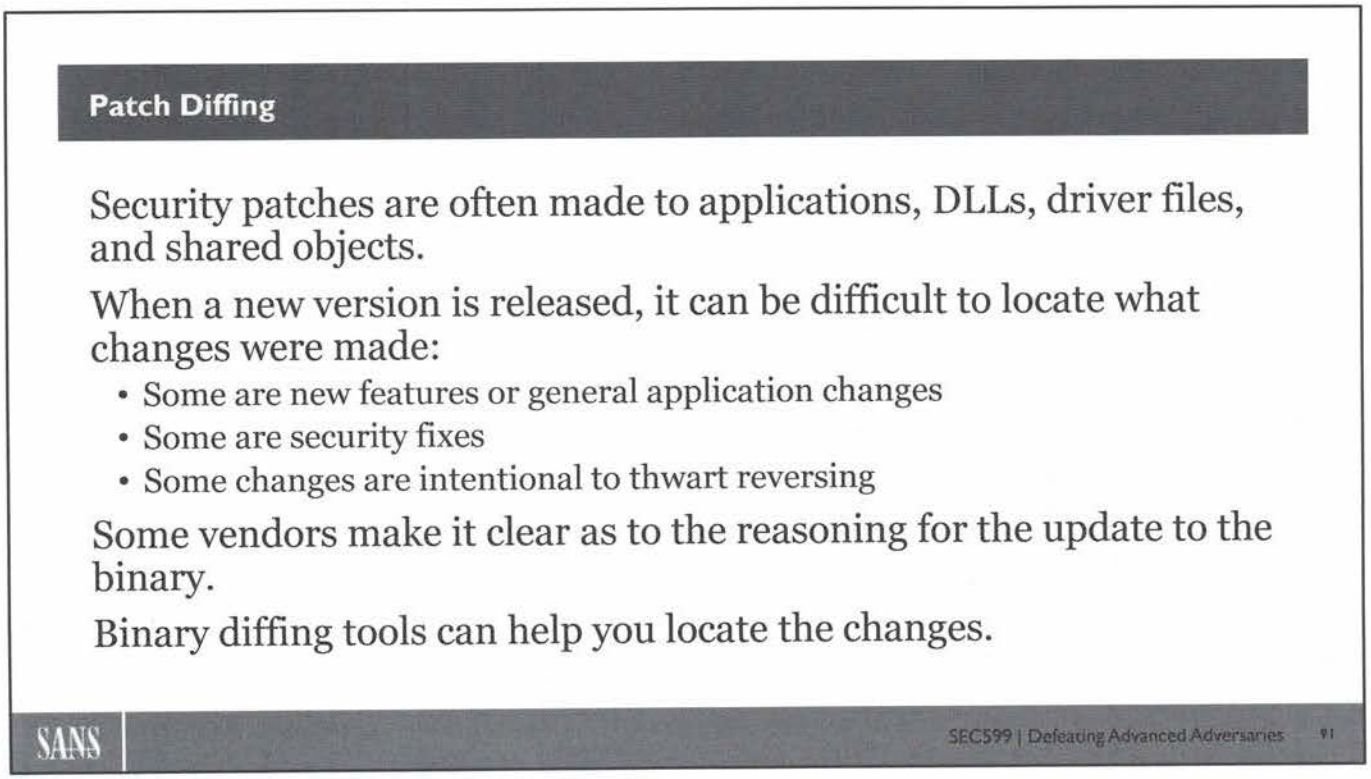

**Patch Extraction Results**

This slide simply shows the results in the remaining folders. These should be easily mappable to security advisories on the Microsoft website. Let's try to do one on the next slide.



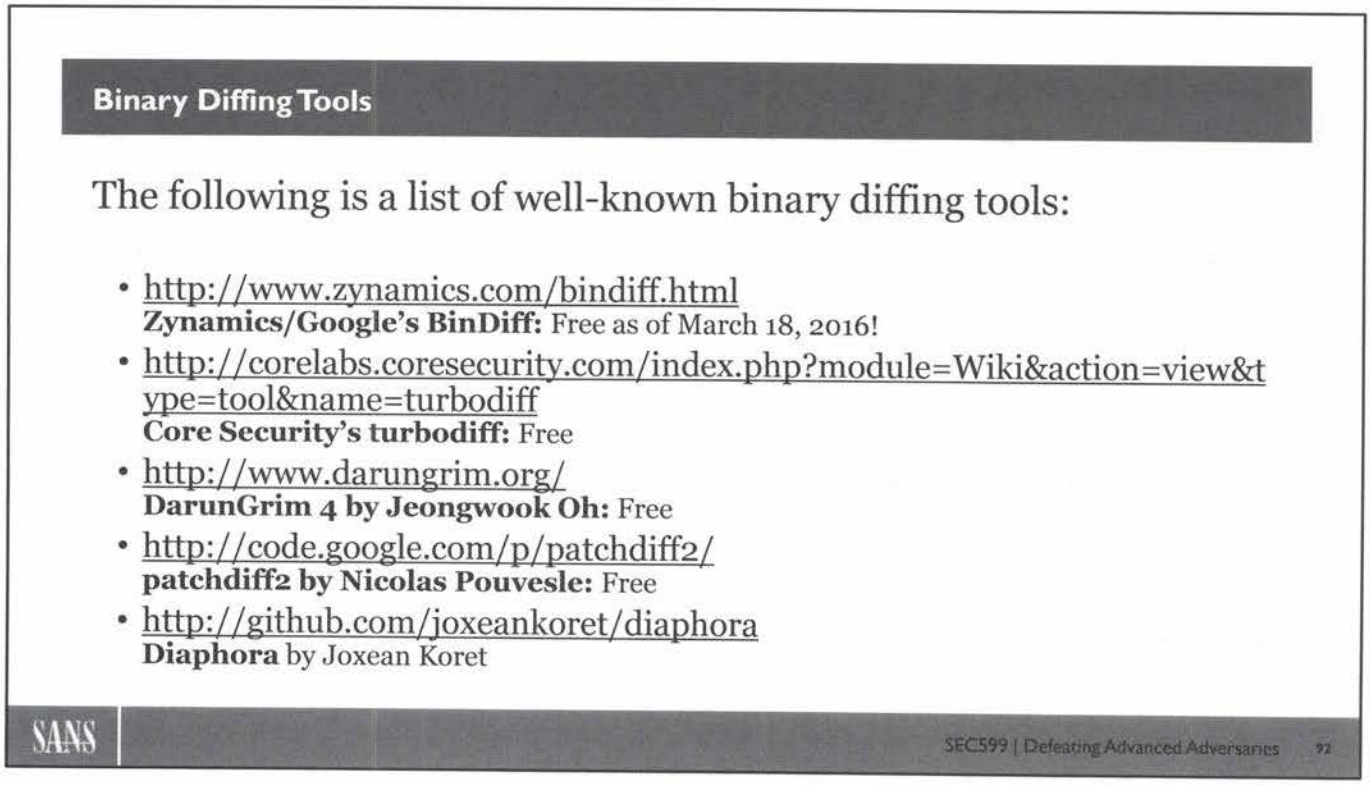

**Mapping a Patched File to the Security Advisory**

When looking at MS17-001 we see that the security bulletin applies to the Microsoft Edge browser. One of the folders, after we ran PatchClean, is "edgehtml.dll." It seems that we were easily able to correlate the patch to the advisory and would be able to continue analyzing.




**Patch Diffing**

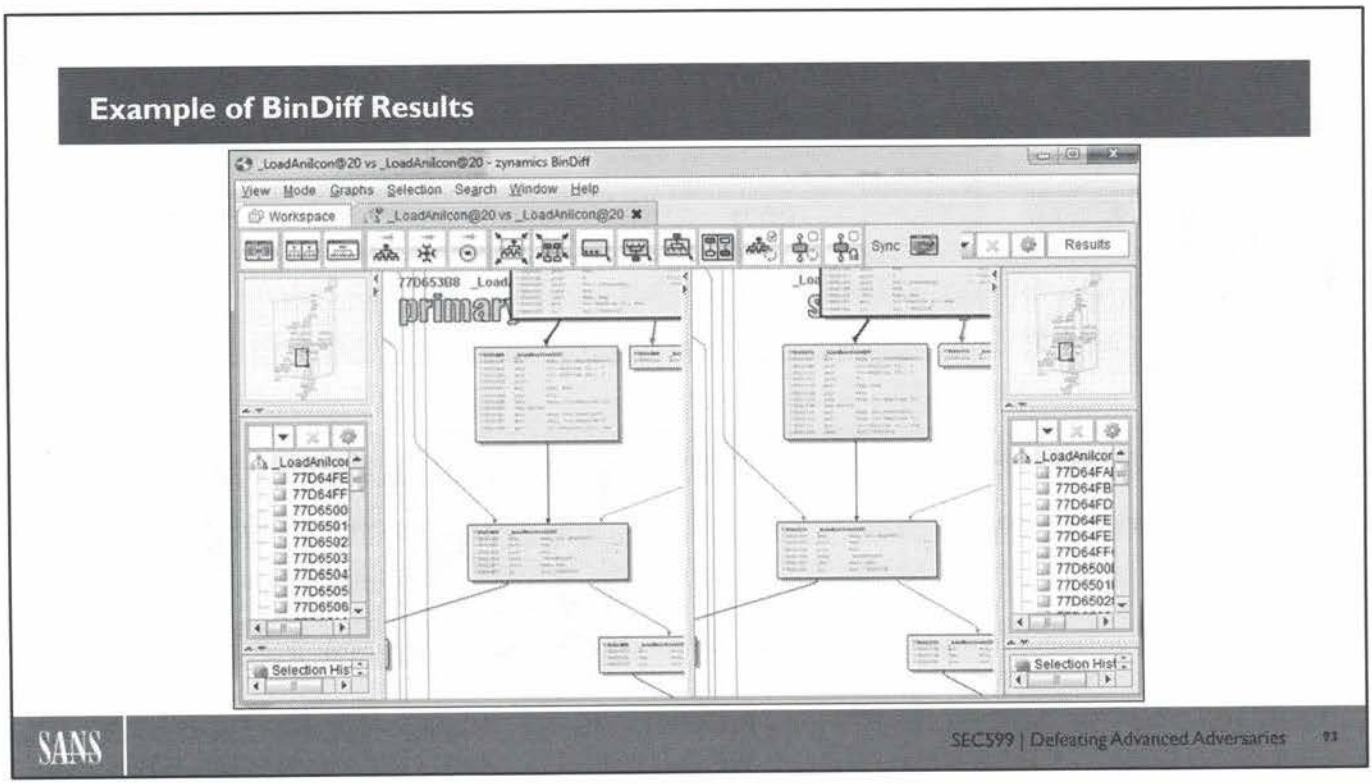
As we are all aware, new versions of applications come out all the time, as do patches to existing DLLs, drivers, and shared objects. Some of these changes are simply new features rolled out or fixes to performance problems. Other changes are vulnerability patches that are certainly of interest. If someone can take the unpatched version of a binary and diff it against the patched version, the code changes may become visible, shining a light on an otherwise unknown vulnerability. Those systems that are properly patched would be safe, leaving anyone who has not patched their system exposed to a potential 1-day exploit. Some vendors make it clear as to the reasoning behind an update, whereas others attempt to hide their intentions. Either way, binary diffing tools can often help us locate code changes that could potentially reveal the patched vulnerability. This is a lucrative practice because many organizations do not patch their systems quickly.




**Binary Diffing Tools**

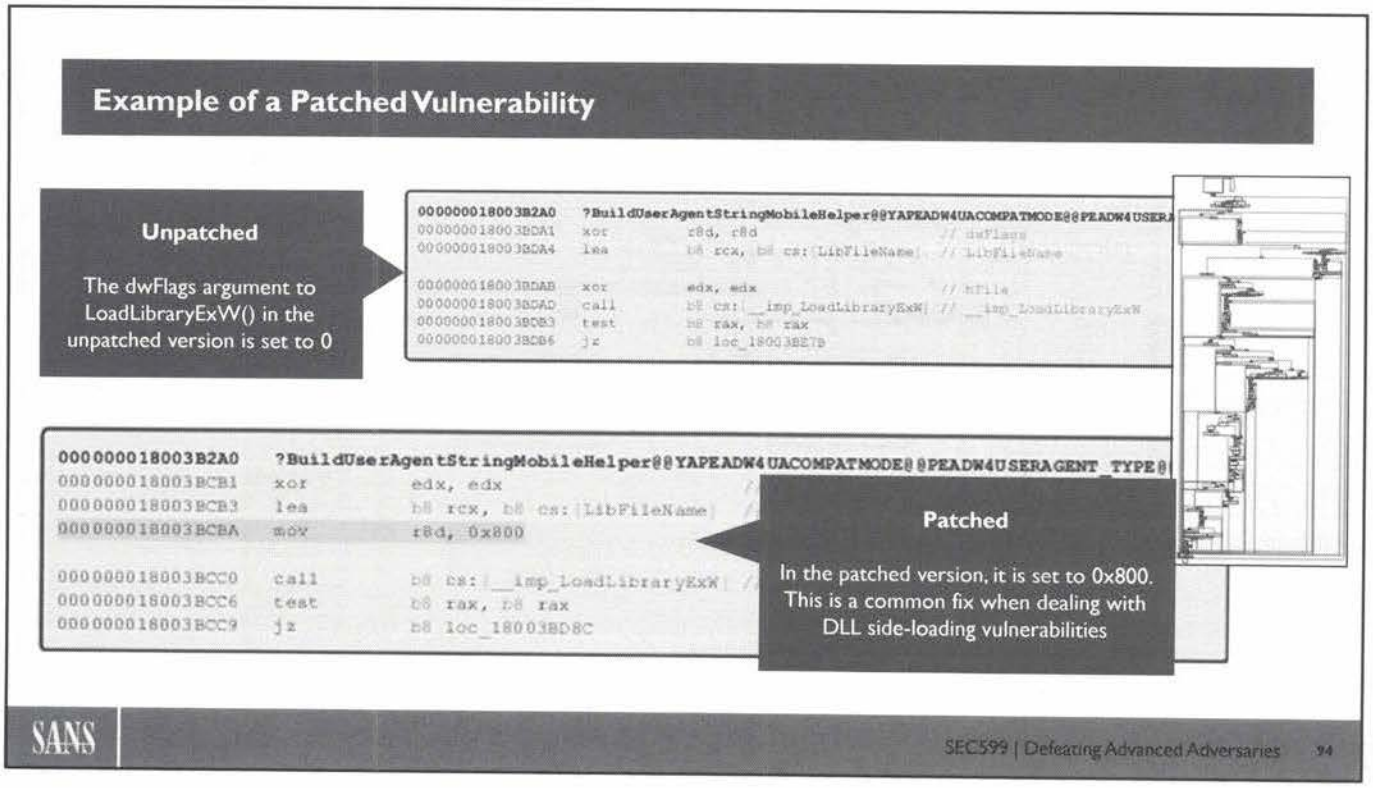
There a few well-known binary diffing tools, most of them free; although many have specific dependencies on versions of IDA.

- **BinDiff**: Created by Zynamics, acquired by Google in 2011 – http://www.zynamics.com/bindiff.html
- **Turbodiff**: Created by Core Security – http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=turbodiff
- **DarunGrim 4**: Written by Jeongwook Oh – http://www.darungrim.org/
- **Patchdiff2**: Written by Nicolas Pouvesle – http://code.google.com/p/patchdiff2/
- **Diaphora:** Written by Joxean Koret – http://github.com/joxeankoret/diaphora




**Example of BinDiff Results**

What you see on this slide is the visual diff of two versions of the same file. On the right, would be the patched version and on the left, is the version from the last time it was patched. When comparing these two versions together the only changes should be related to the most recent security fix. Some DLL's and other binaries might have over 20,000 functions. Using these diffing tools, we can greatly reduce the number of files we would have look at when trying to identify modified code in relation to a vulnerability.



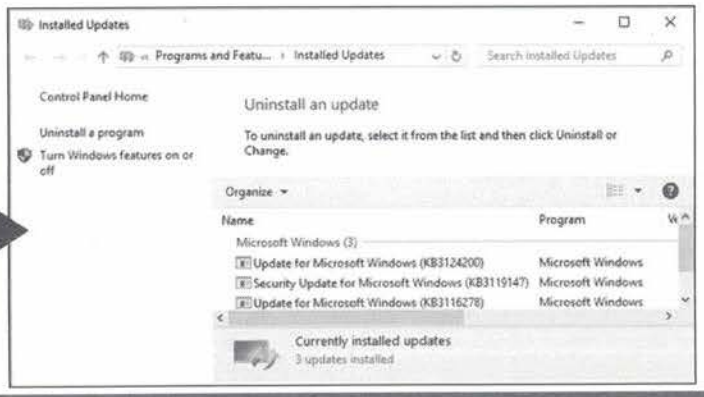

**Example of a Patched Vulnerability**

When running a Visual Diff and analyzing the changes, the block shown on this slide shows the issue. The dwFlags argument to LoadLibraryExW() in the unpatched version is set to 0, while in the patched version it is set to 0x800. This is a common fix when dealing with DLL side-loading vulnerabilities. A dwFlags argument of 0 results in the potential loading of DLL's from outside of safe locations. A dwFlags option of 0x800 states that DLL's may only be loaded from the "%SystemRoot%\Windows\System32" folder.

## Uninstalling a Patch

Sometimes, when testing patches, diffing, testing exploits, etc., you need to uninstall an update.

- Simply go to Control Panel, "View Installed Updates," and double-click on the one to uninstall:




This is an example from a Windows 10 base build with minimal updates, hence the low number listed.



**Uninstalling a Patch**

Sometimes, a patch was already applied to a system you want to test, or you may want to uninstall an update for any number of reasons. The process is simple because Windows archives the old versions of patched DLLs and other files. Simply go to your control panel and type "View Installed Updates" into the search field. A box with the installed updates will appear as shown on the slide. When you find the update you want to uninstall, double-click it, and you will be asked if you are sure you want to uninstall this update. This is an example from a base build with minimal updates, hence the low number listed.

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

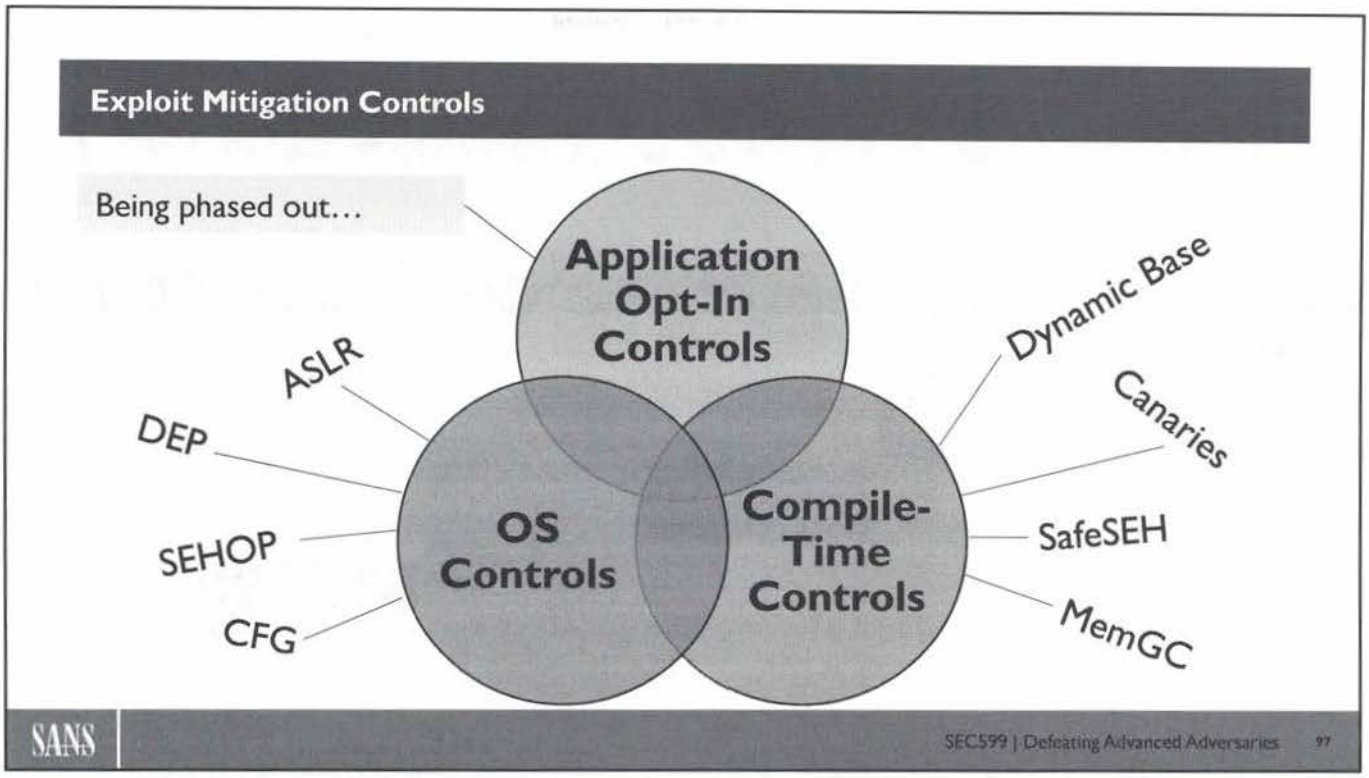
**Securing the Network**
- Network Access Control & 802.1X

**Securing Software**
- Software Development Lifecycle (SDL) & Threat Modeling
- Vulnerability Assessments
- Exercise: Authenticated Scans Using Nessus
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation Using Compile-Time Controls
- Exploit Mitigation Techniques – Exploit Guard, EMET, & Others
- Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**
- OS Hardening & Best Practices
- Endpoint Protection Solutions
- Application Whitelisting to Stop Payload Execution
- Exercise: Configuring AppLocker



This page intentionally left blank.

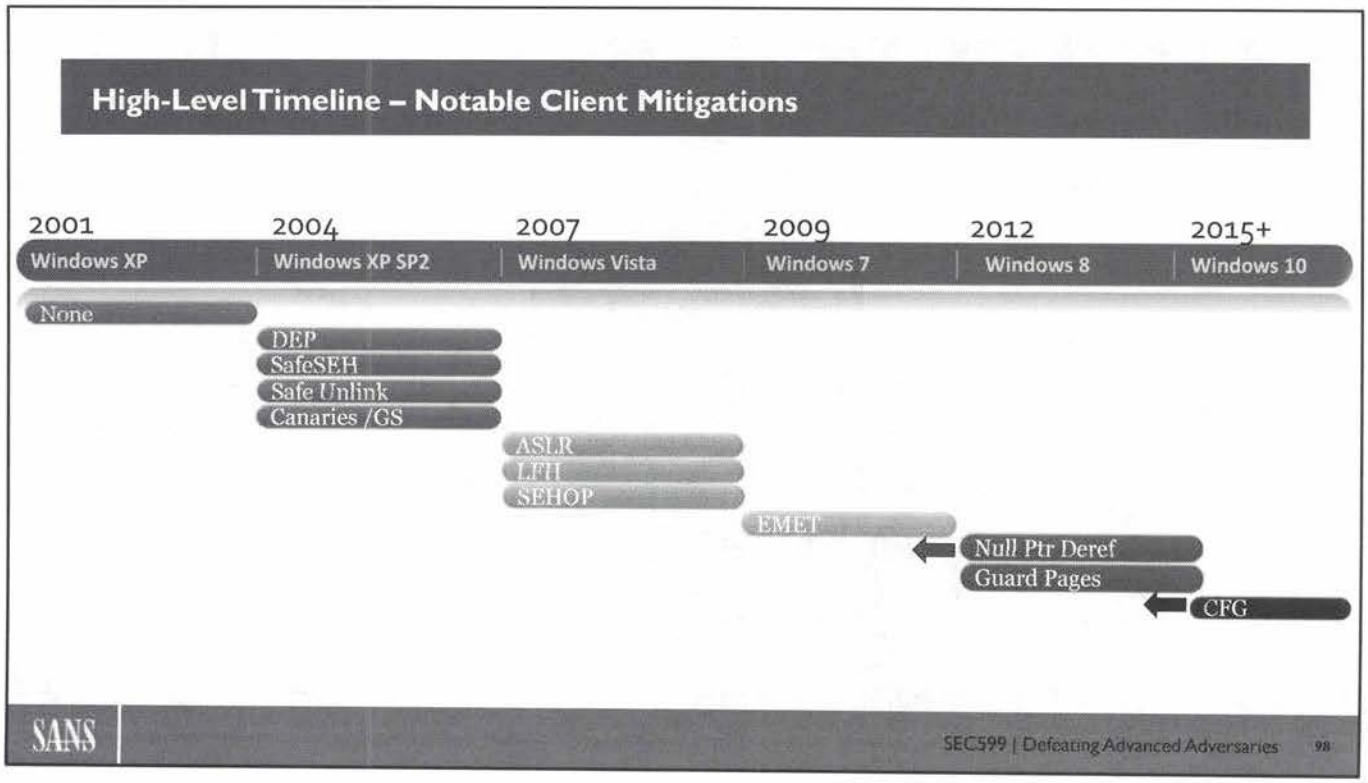



**Exploit Mitigation Controls**

First, let's briefly discuss the role of exploit mitigations. We are all aware of the concept of "Defense in Depth." The idea is that any one control may fail so we want as many as possible without impacting application or system performance too significantly. If we only utilize a single control such as Data Execution Prevention (DEP) and an attacker figures out a way to disable it, then there is nothing left protecting the application or system from compromise. By layering on various controls, it can stop or at least greatly increase the difficulty to achieve exploitation.

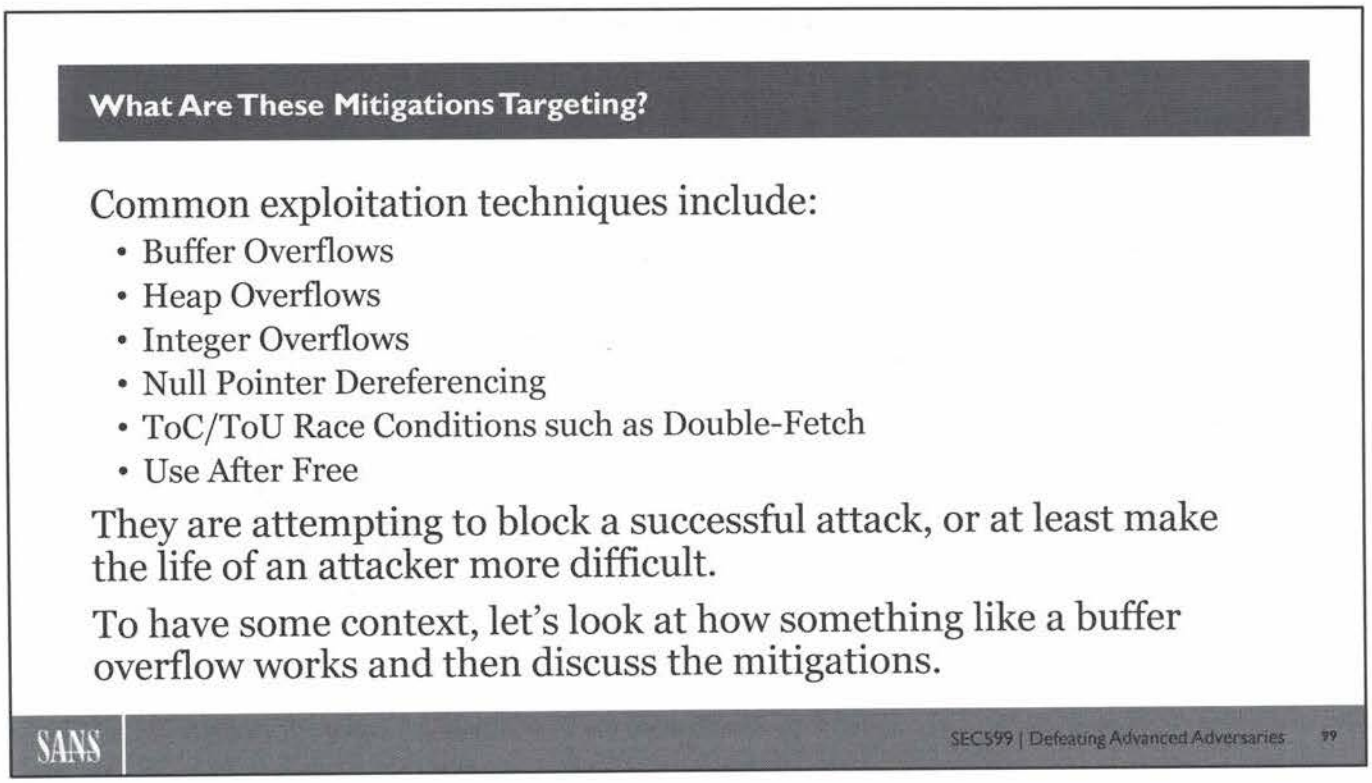
The basic Venn diagram on the slide shows three categories of exploit mitigation. On the top is "Application Opt-In Controls." This is the least effective category as typically a flag exists in the compiled program and at runtime, a control is enabled or disabled based on this flag. An attacker could theoretically utilize a hex editor to modify these flags if they can determine where it is stored, disabling the protection. Let's use the good example of DEP. On Windows, DEP is controlled through the system control panel. An administrator can turn DEP on or off for all applications, or define exceptions. You wouldn't also want the application itself to decide if it wants to opt-out and ignore the settings in the control panel. As an example to demonstrate the issue, up until Windows 7 there was an API that existed in ntdll.dll called the "LDRPCheckNXCompatibility routine." It checks the application to see if it is to participate in DEP. It was removed as disabling DEP was simple and possible by replaying runtime code in ntdll.dll during an exploit. You can find out more about this technique here: http://uninformed.org/index.cgi?v=2&a=4&t=txt

The other two categories, "OS Controls" and "Compile-Time Controls" provide the best options. OS controls include protections such as Address Space Layout Randomization (ASLR), DEP, Structured Exception Handling Overwrite Protection (SEHOP), and Control Flow Guard (CFG). The operating system must support these controls, and sometimes even the hardware. Each OS is different, but they are typically designed to be controls that cannot be turned off by an application. They are system enforced. Compile-time controls are exactly how they sound; they are controls that are added during compile time. These often insert code or metadata into the program. Examples include stack and heap canaries, MemGC, SafeSEH, and Dynamic Base. We will discuss a sample of the most prominent controls in this module. As we increase the number of controls and move into the merged areas of the circle our protection should increase.



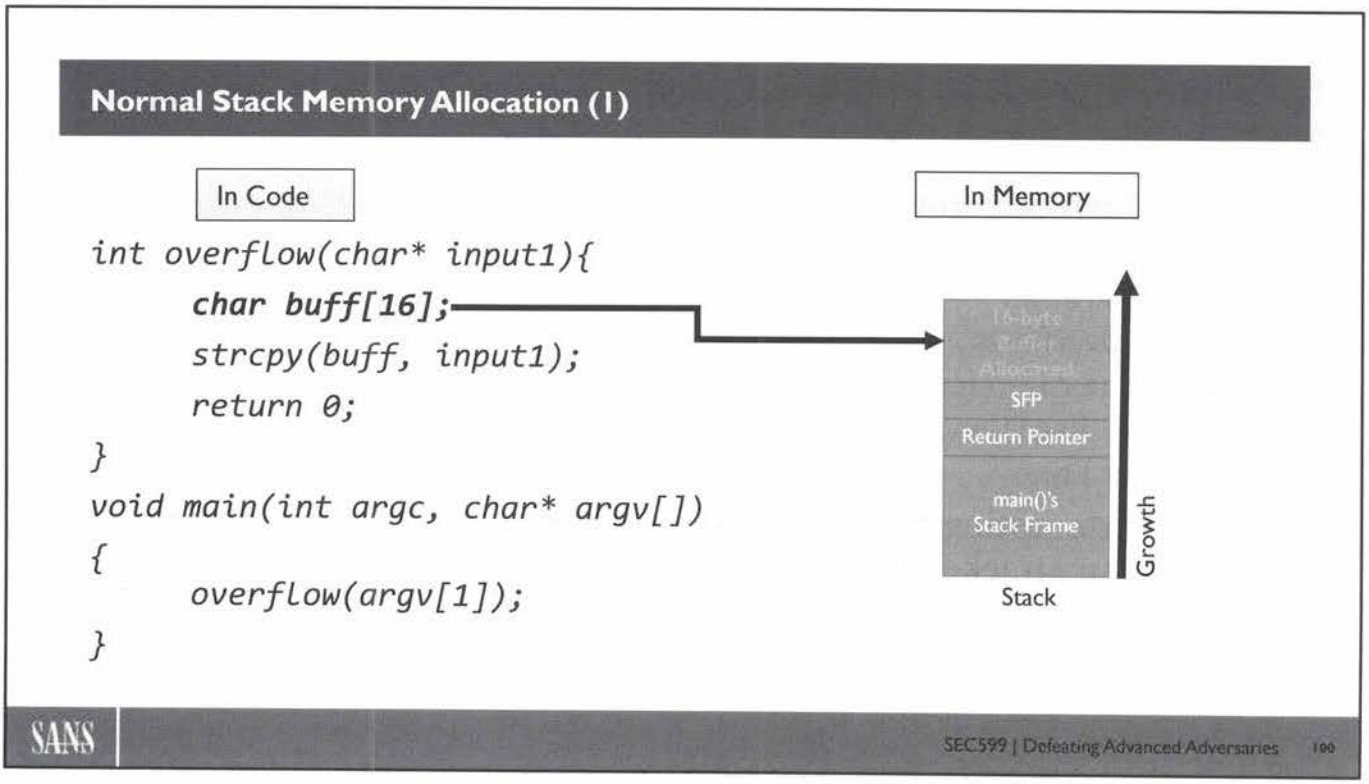

**High-Level Timeline – Notable Client Mitigations**

This slide shows a high-level timeline of exploit mitigations added or made available over the years. This is not conclusive by any means, and we will address many of them in this module.




**What Are These Mitigations Targeting?**

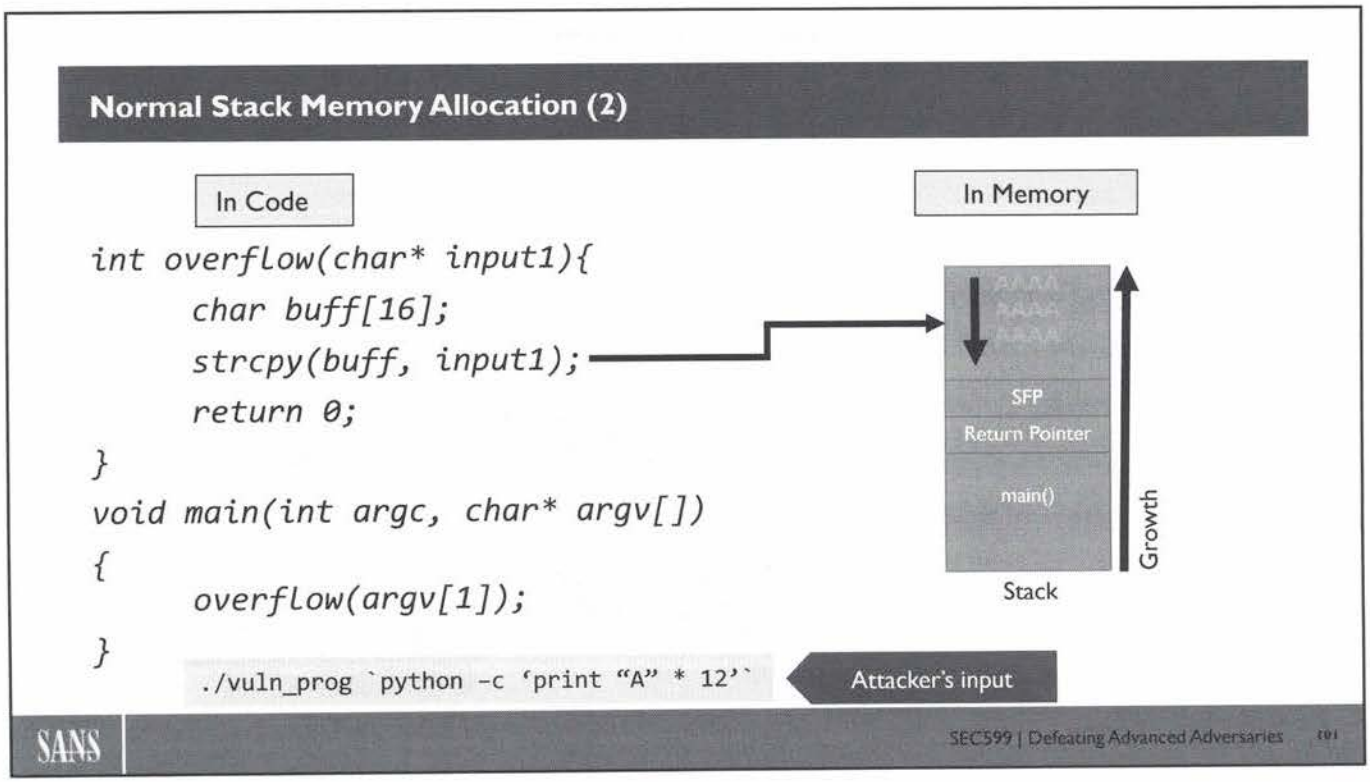
There are many different bug classes, each varying in difficulty in relation to discovery, exploitability, reliability, etc.... Some common bug classes related to low-level languages such as C and C++ include buffer overflows, heap overflows, integer overflows, null pointer dereferencing, race conditions, and use after free. The exploit mitigations that we are discussing focus their efforts on preventing the exploitability of these vulnerabilities. The majority of operating systems and large applications are written in C and C++, as well as assembly, and Objective-C. The main benefits of using low-level languages are their ability to directly access memory, processor registers, and other low-level functionality. This access also allows for costly mistakes. Let's look take a look at a basic buffer overflow that will provide context when discussing the mitigations.




**Normal Stack Memory Allocation (1)**
On the left of this slide is the C source code:

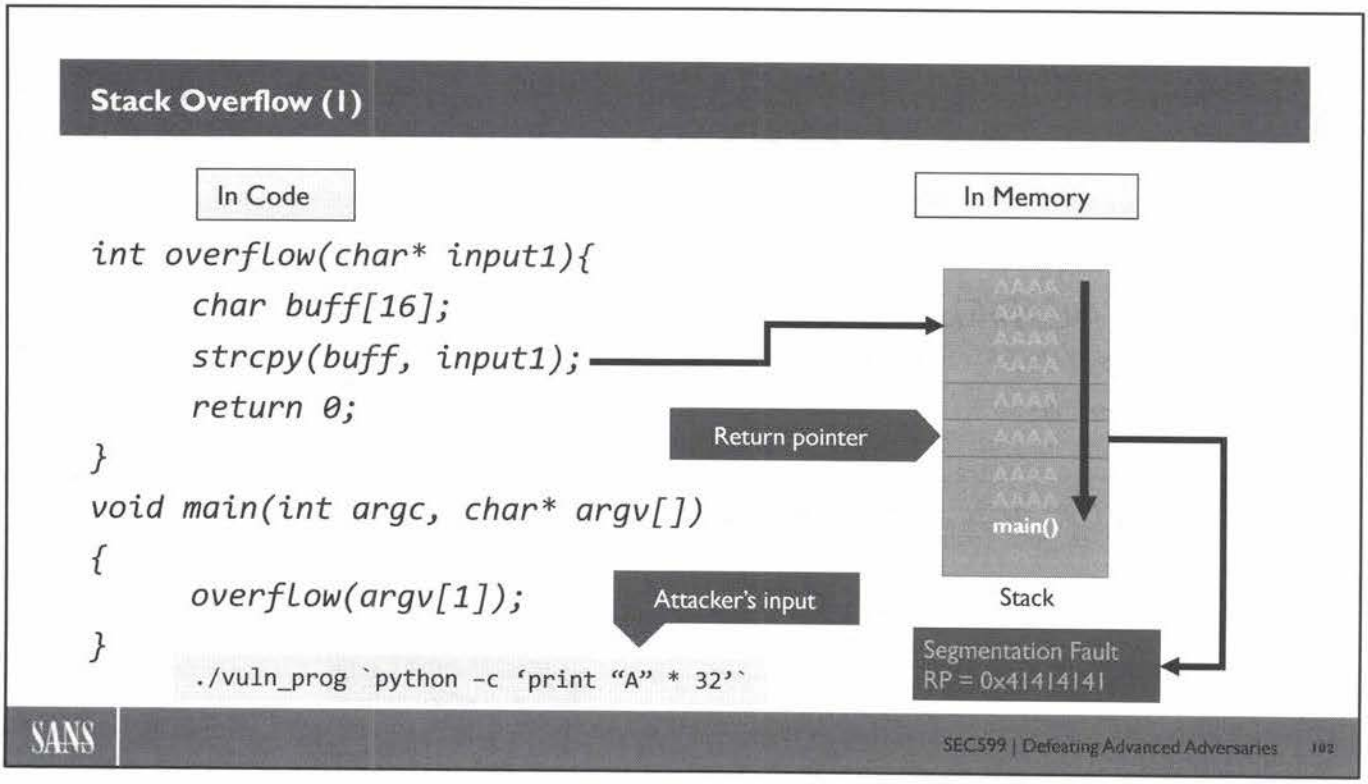
```
int overflow(char* input1){                    // This is a function called overflow() which contains a
buffer overflow
   char buff[16];                  // Allocating a 16-byte buffer called buff
   strcpy(buff, input1);                       // The vulnerable strcpy() function copying user-supplied
data to the buffer
   return 0;                                   // Returning a status of 0 if all is find
}
void main(int argc, char* argv[])  // This is the main() function with which all C programs begin.
{
   overflow(argv[1]);                          // The main() function calls the vulnerable overflow()
function, passing stdin
}
```

On the right is the stack of the process once the program is compiled and run. The stack is a region in memory that stores data associated with function calls. Each function called gets its own allocation of memory on the stack called a stack frame. The stack grows from high memory towards low memory. If function A() calls function B(), then function B()'s stack frame will be built on top of function A()'s. As the code in a function finishes, control is passed back to the calling function via the return pointer. In the image on the right, you can see that main()'s stack frame is at the "bottom of the stack", on top of which stack frames for other functions (e.g. overflow) are built.




**Normal Stack Memory Allocation (2)**

On this slide, you can see that the attacker is using Python to send 12 A's as input into the program. This input goes into the program via standard-in (stdin). "Argv" is the argument vector. At argv[0] is the name of the program and argv[1] would be the first argument passed by the attacker, which is the 12 A's. If a second argument was passed it would be reachable at argv[2]. Again, it is simply the argument vector. In the main() function the overflow() function is called, and passed argv[1] as an argument. The overflow function takes a pointer "char* input1" to the argument and names it input1. The 16-byte buffer is then allocated, and then strcpy() is called, copying the 12 A's into the 16-byte buffer. The problem with strcpy() is that it does not provide any bounds checking. In other words, there is no size argument to limit the amount of data copied into the allocated buffer. So, if the attacker passed in 100 A's strcpy() would happily copy all of it into the buffer, overwriting important items such as the return pointer back to the main() function.




**Stack Overflow (1)**

On this slide, the attacker has sent in 32 A's instead of only 12. As you can see, strcpy() happily wrote past the 16-byte buffer allocation, overwriting the return pointer back to main() amongst other things. When the process went to return control to main it returned to "AAAA." The letter "A" in hex-ASCII is "0x41." The "0x" on the front of the number indicates that it is a hexadecimal value or Base-16. So "AAAA" maps to "0x41414141" which is why we are seeing that show up in the segmentation fault at the bottom. The attacker now knows they can gain control of the program. At this point, they would use a debugger or disassembler to determine the exact number of bytes before reaching the return pointer.

## Stack Overflow (2)

1. Using a debugger, the attacker gets the static memory address of the buffer.
2. They place their shellcode into the buffer and overwrite the return pointer with the address of the buffer.

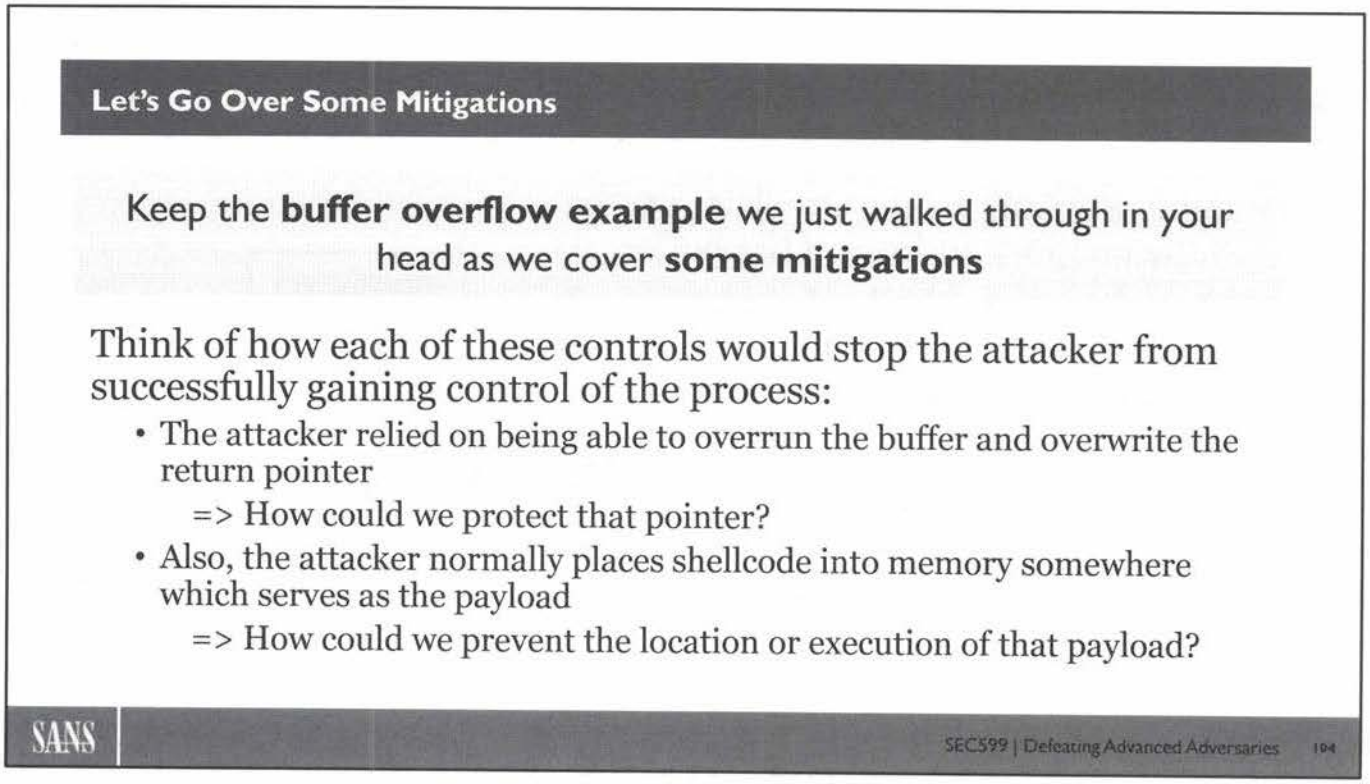
Segmentation Fault
RP = Address of Shellcode

3. When the process goes to return control to main(), control is instead passed to the attacker's shellcode.




**Stack Overflow (2)**

Once the attacker determines the exact number of bytes to get to the return pointer, as well as the address of the buffer, they can insert their shellcode into the buffer first, and then overwrite the return pointer back to main() with the address of their shellcode. The exploit mitigations will focus on things like randomizing the addressing in memory, preventing execution in writable regions, place guards into memory, and many other approaches.




**Let's Go Over Some Mitigations**

Now, let's keep the buffer overflow example we just discussed in our mind, as we cover some mitigating controls. Try to think how each of the introduced controls could possibly stop the attacker from successfully gaining control of the process (& thus being able to execute malicious code).

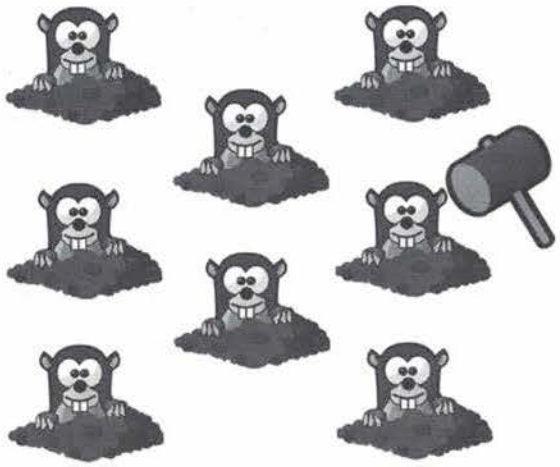
There are two interesting things to note:

1. The attacker relied on being able to overrun the buffer and overwrite the return pointer. How could we protect that pointer?
2. Furthermore, the attacker usually attempts to place shellcode into memory somewhere, which will serve as a payload that is to be executed. How could we prevent the location or execution of that payload?

The author recognizes this is not the easiest topic of the day, but if we want to protect against advanced adversaries, it's vital we understand how modern exploit mitigation strategies work. As always, should you have questions or would look some additional guidance on this subject, please don't hesitate to contact your instructor or TA. Let's have a look!

## Address Space Layout Randomization (ASLR)

- Take away predictability by randomizing regions of memory each time a process is started.
- On some OSes, this includes libraries, while others do this separately as a compiler option.
- Attackers like to rely on the static locations of objects.
- Increasing the entropy increases the difficulty of exploitation.



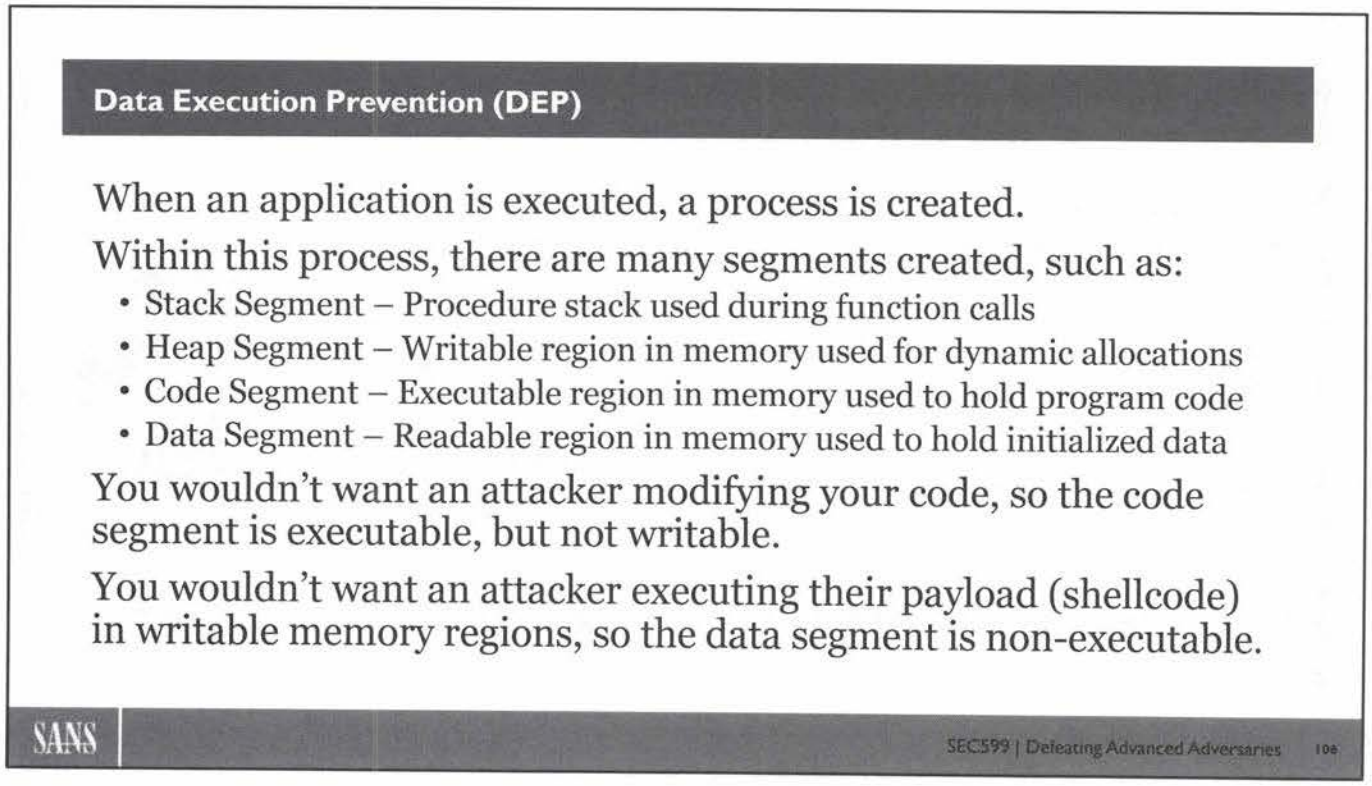


**Address Space Layout Randomization (ASLR)**

Imagine if each time you went to retrieve the salt and pepper from your kitchen that it was in a different location. Even though the last time you used it, you put it in a specific location for retrieval later, the next time you went to use it the location changed again. This would likely cause frustration. The concept is not so different from ASLR. Attacker's like for things to be static in memory no matter on what system the vulnerable application is running. This allows for them to use static addressing in their exploits. If an attacker gains control of a process and attempts to execute their payload, but the location has changed, it will likely crash and fail.

On some operating systems, the libraries are randomized as part of the overall system setting for ASLR. Many Linux variants include a file called "randomize_va_space" which stores a value of 0, 1, or 2. If the value stored is a 0 then ASLR is off, including libraries. If the value is a 1 or a 2 then ASLR is on. On the Windows OS ASLR cannot be turned off for the system; however, there is a compile-time control known as "DynamicBase" which determines if a Dynamic Link Library (DLL) is to be randomized once loaded into a process.

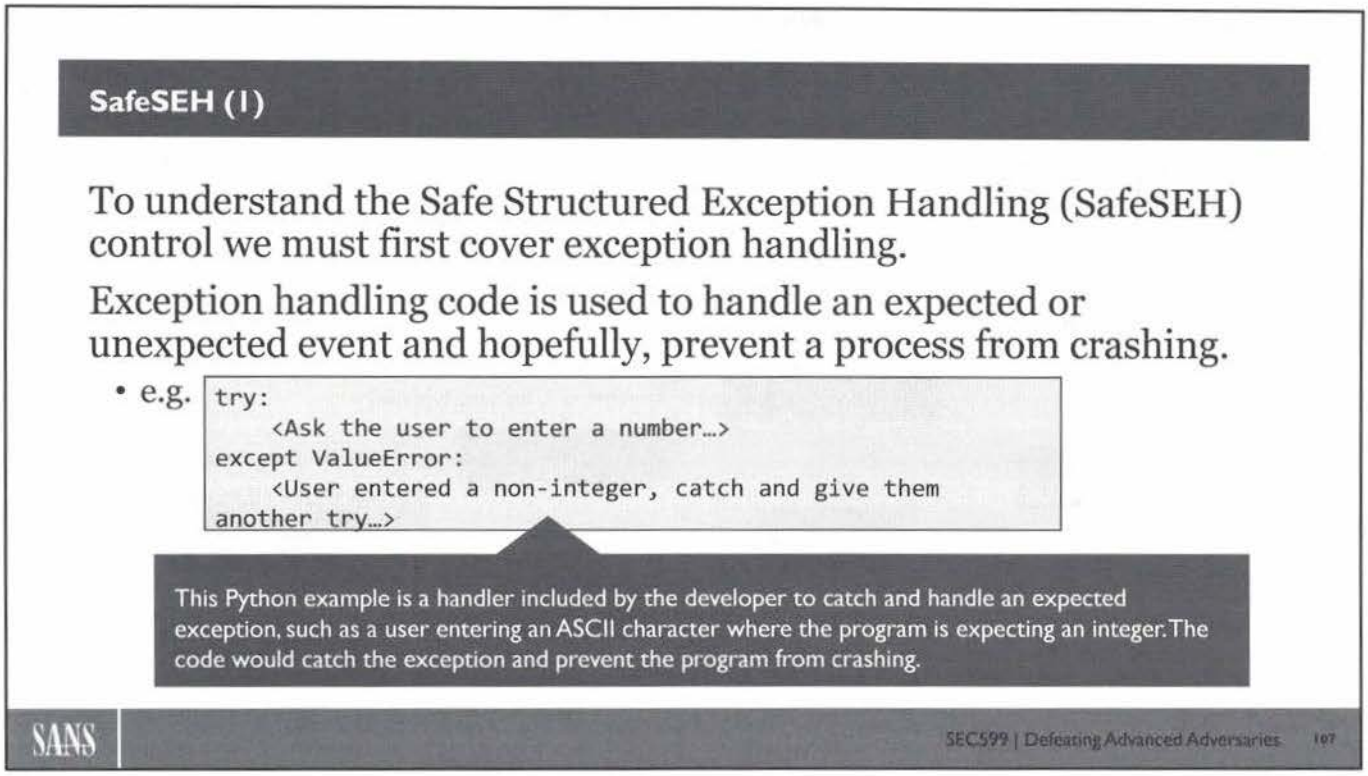
Take the earlier example of the salt and pepper. If your kitchen is rather small, and you know that the salt and pepper must be stored somewhere in the kitchen, then the chances of you guessing the right spot are pretty good in a short number of tries. At least when comparing it to an example where your kitchen is the size of a sports stadium. If this were true the chances of successfully guessing the right location of the salt and pepper is greatly lessened.




**Data Execution Prevention (DEP)**

DEP is a mature OS control that is quite simple to explain. Executable code resides in its own segment in a process called the code segment. You certainly wouldn't want anyone to be able to modify the code in your program. When thinking about the three basic permissions, read/write/execute, then it is easy to see why the code segment should be executable but not writable. When dealing with writable segments of memory within a process such as the stack and heap it is clear that they need to be writable. Now since those regions are writable we have concerns around an attacker inserting their own malicious code into these areas and somehow causing code execution to occur. We can mitigate this concern by marking this memory region as writable, but not executable.

DEP must be supported by the processor. In RAM, when an allocation occurs, a unit of measurement known as a page is used. On most OSes, a page of memory is 4KB for alignment purposes. They are allocated by the processor. As pages of memory are allocated a special bit is set, known as the eXecute Disable (XD) or No eXecute (NX) bit, depending on the hardware architecture. Regardless, they are identical as to their role. The bit simply determines if the page is writable or executable, as the idea behind the control is that you cannot be both.

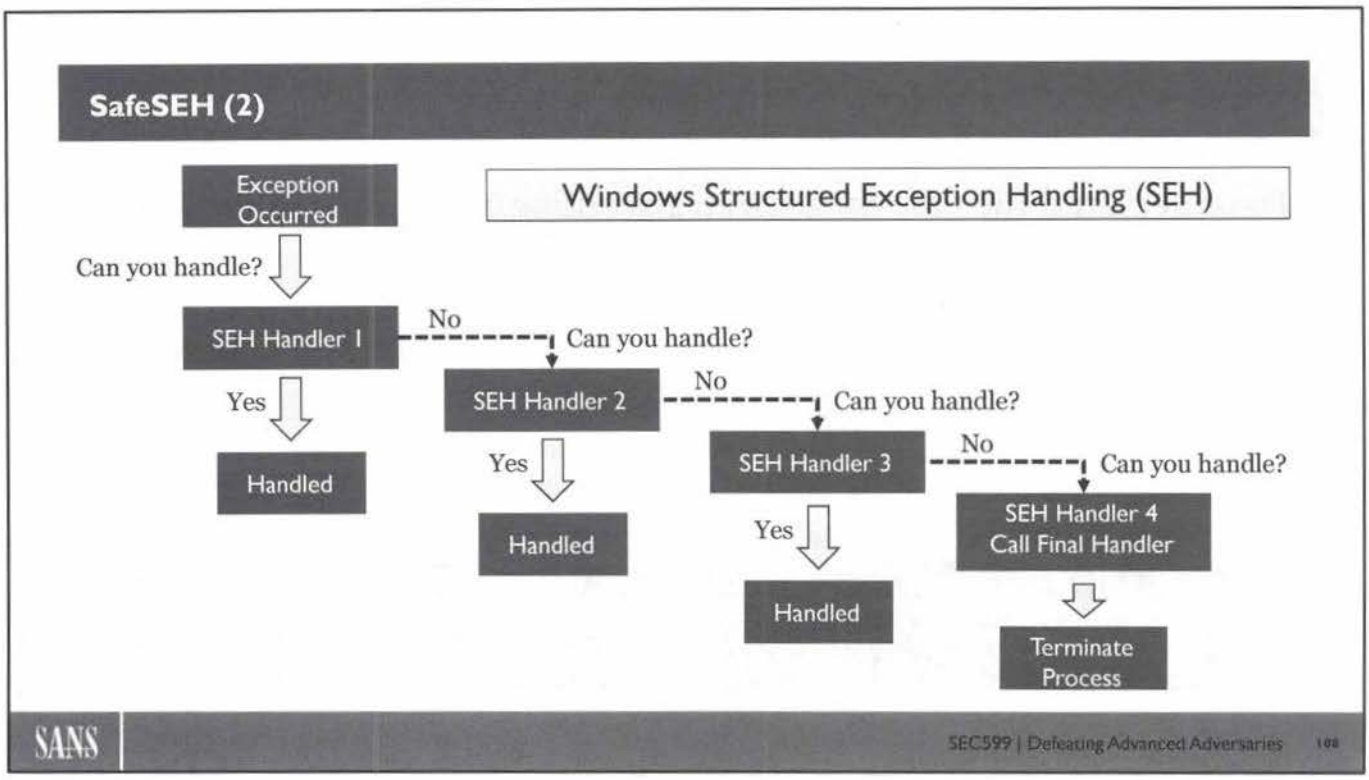



**SafeSEH (1)**

To explain the Windows Safe Structured Exception Handling (SafeSEH) compile-time exploit mitigation control we must first cover basic exception handling and Windows SEH in general. An exception is something that occurs within a process, such as an anomaly, or an intentional attempt to cause program fault, that potentially affects the stability of the process. If we have exception handling code within the program then the exception that has occurred can potentially be handled, preventing the process from crashing. Some exceptions can be anticipated, while others are unexpected. Look at the following Python-like example:

**try:**
**<Ask the user to enter a number…>**
**except ValueError:**
**<User entered a non-integer, catch and give them another try…>**

The "try" and "except" syntax is used to create a handler in Python. We are basically saying that we wish to try the following code, but if an exception occurs we wish to have it handled. We are asking the user of the script to enter in an integer value. If the user enters anything other than an integer value, Python will throw a "ValueError" exception, which can be anticipated. When this type of exception is experienced our code would catch it and pass control back to the "try" block again. Handlers are supported by almost all programming languages.



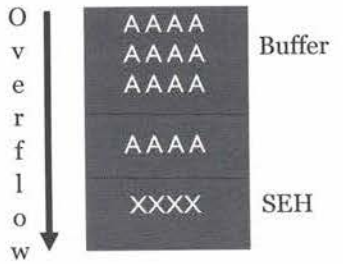

**SafeSEH (2)**

The windows SEH mechanism is used to handle various types of faults that cannot or are not handled by developer code. An example of an exception that would cause control to shift the SEH mechanism is when the processor attempts to read from or write to a memory address that is not mapped into the process. Processes only take up the physical memory and virtual addressing that is required to run the program. If they run out of memory, more can be requested. If something causes a process to attempt to read or write to an address that is not mapped from virtual memory to physical RAM then an exception occurs. Control would move to the SEH chain. It is called a chain because, as you can see on the slide, if one exception handler cannot handle the exception, control is passed down the chain until it is handled or it reaches the end. If the end is reached and none of the handlers were able to handle the exception, then the program is terminated. The handler code resides in various DLL's such as ntdll.dll. The pointers to or addresses of the handlers are stored on the procedure stack for the thread.

## SafeSEH (3)

SafeSEH is an optional compile-time control that builds a table of all valid handlers in a DLL; the table stores the addresses of each valid handler.

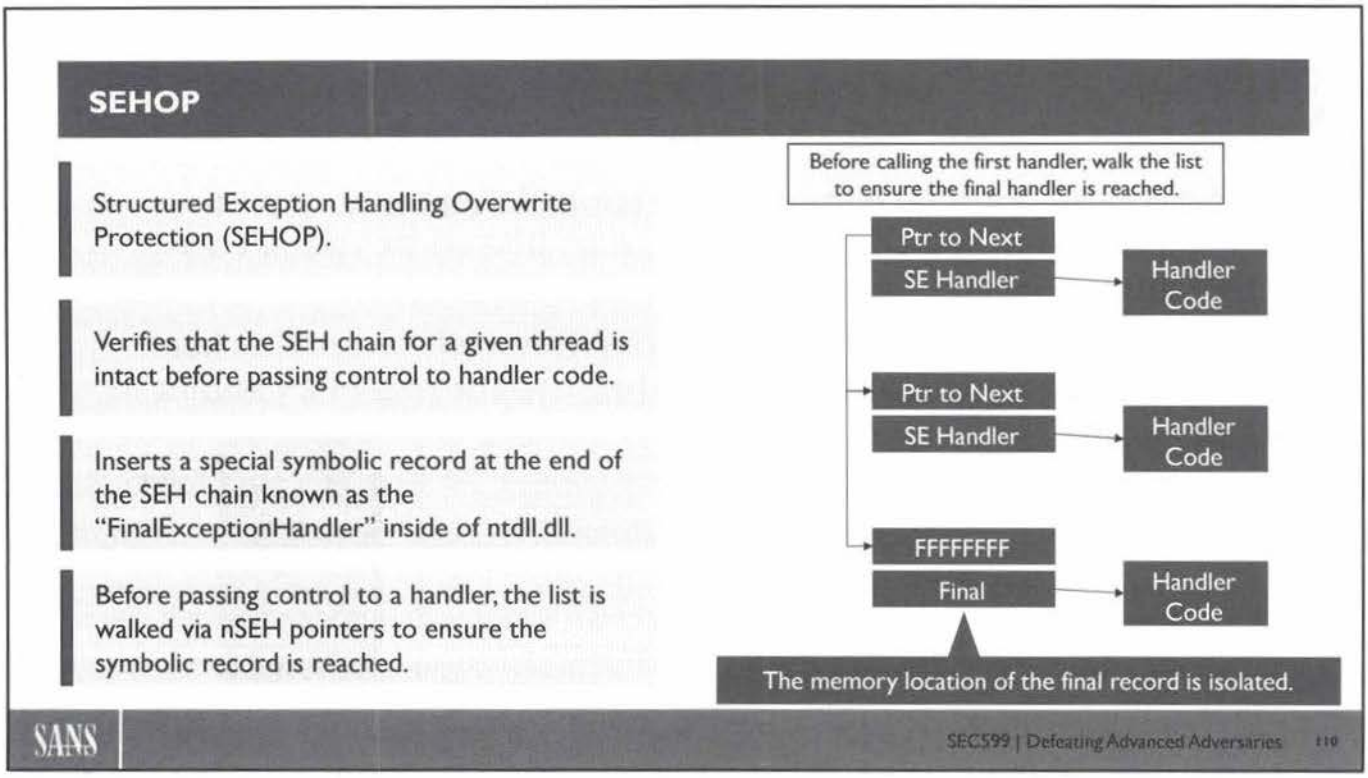
SafeSEH is aimed at stopping buffer overflows where an attacker attempts to overrun a buffer, overwriting the address of a handler to hijack control.






**SafeSEH (3)**

SafeSEH is a compile-time control that builds a table of all valid handlers inside of a DLL. Each handler has a starting memory address within a module. This is the address stored in the SafeSEH table. A common attack technique is for attackers to overwrite the location in memory where the address of a handler is stored. In case you are curious, the SEH chain resides on the stack for each thread within a process. An attacker would overwrite the handler address in memory with the address of their choosing. There are common techniques used by attackers to gain control of a process via SEH overwrites. If an attacker overwrites one of these SEH addresses and the address written by the attacker points into a SafeSEH-protected DLL, they would be caught and the process terminated. It is not seen as a very effective control as all modules (DLL's) loaded into the process must participate in the control. A single module that does not participate in the control will render this exploit mitigation worthless.



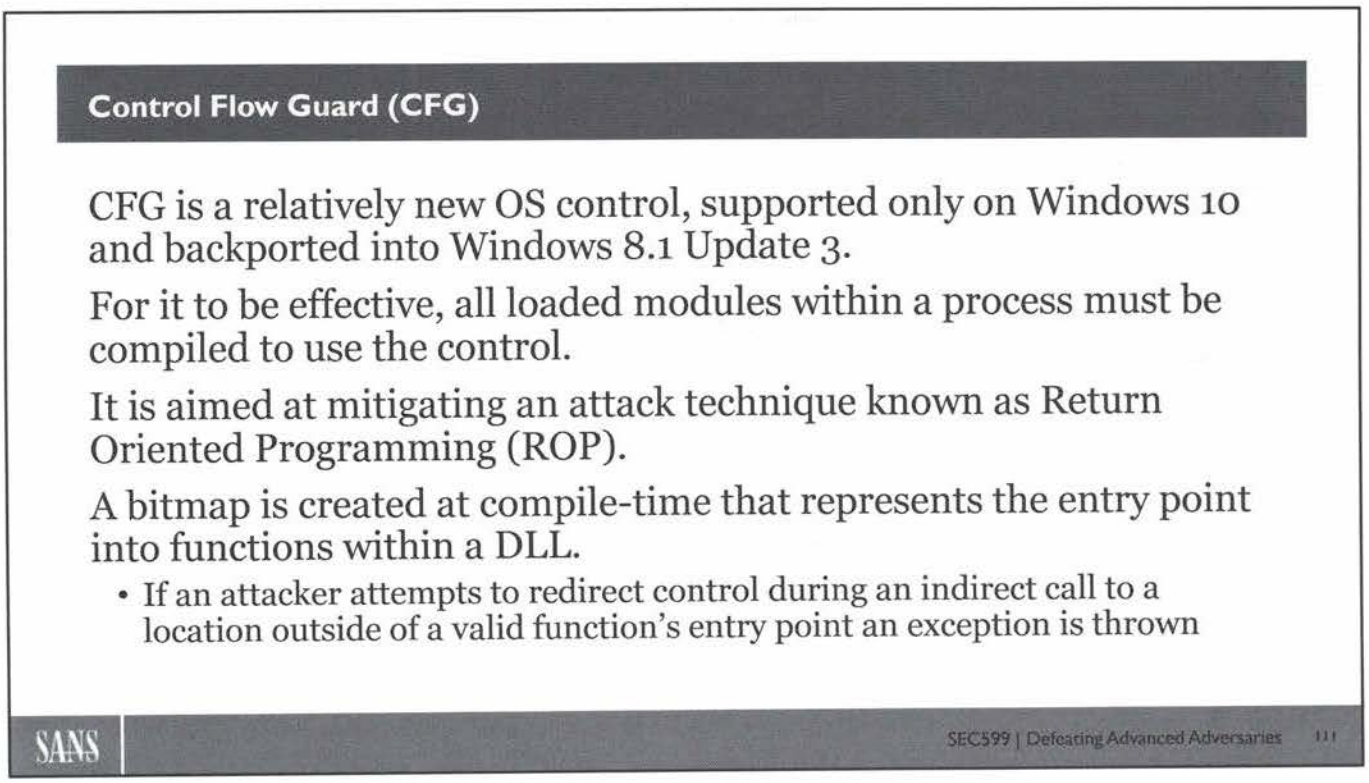

**SEHOP**

The Structured Exception Handler Overwrite Protection (SEHOP) control was added into Server 2008 and Vista; however, it is disabled by default on almost all versions of Windows. This is due to the potential lack of application support for the protection, although it can be enabled, typically through the use of Microsoft's Enhanced Mitigation Experience Toolkit (EMET). Normally, if you follow the nSEH pointers down the stack, you will reach the end of the list. If a handler has been overwritten, it is likely that walking the pointers will no longer reach the end of the list.

As described by Matt Miller (Skape) at Microsoft, the SEHOP control works by inserting a special symbolic record at the end of the SEH chain. Prior to handing control off to a called handler, the list is walked to ensure that the symbolic record is reachable.

References:

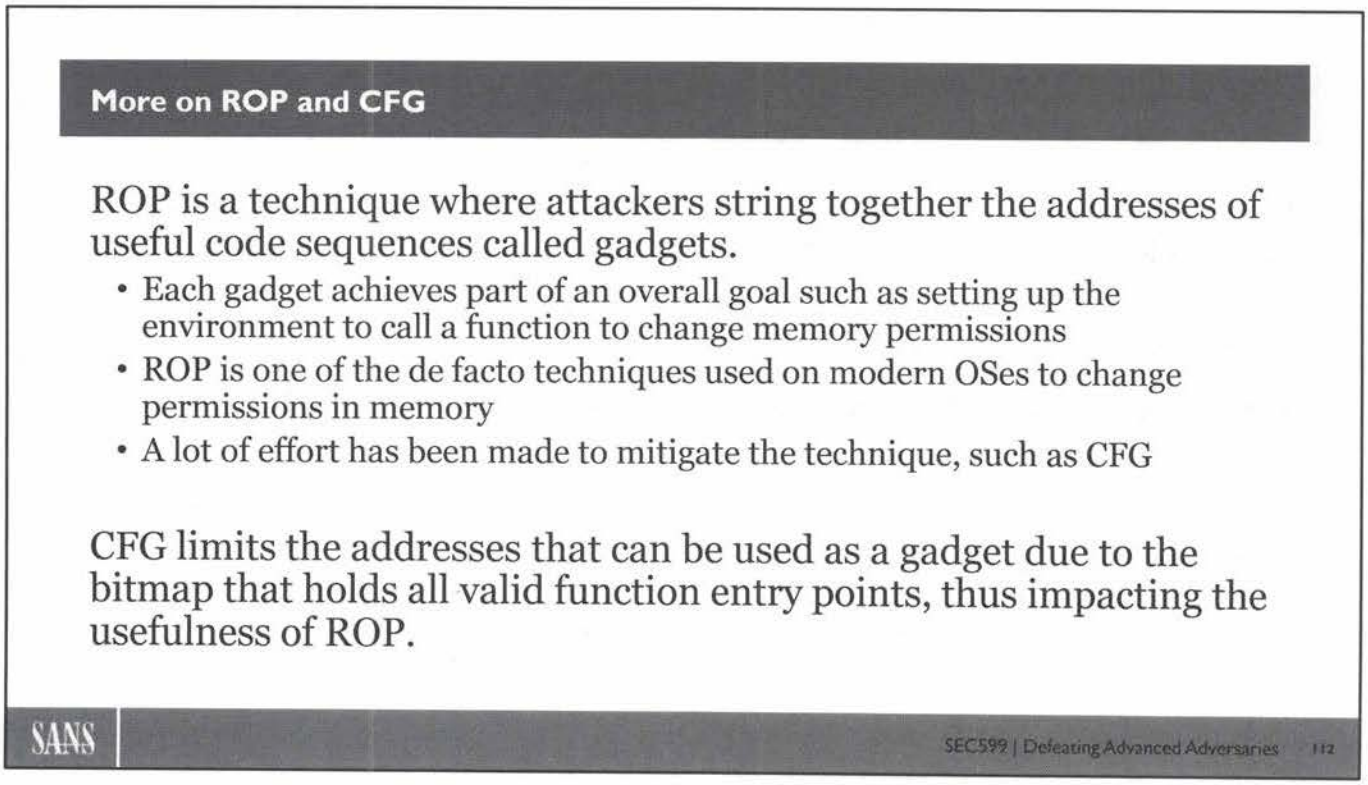
Miller, Matt (2009-02-2). Preventing the Exploitation of Structured Exception Handler (SEH) Overwrites with SEHOP. Retrieved January 11, 2017, from technet.Microsoft.com Website http://blogs.technet.com/b/srd/archive/2009/02/02/preventing-the-exploitation-of-seh-overwrites-with-sehop.aspx




**Control Flow Guard (CFG)**
CFG is a newer control that requires support by the OS and also that each module (DLL) be compiled with the control as it requires code insertion. It is supported on Windows 10 and was backported into Windows 8.1 Update 3, and is also supported in Server 2016. Again, aside from the OS support requirement, CFG is only effective if all loaded modules (DLL's) are compiled to support the control. The control is aimed at mitigating a common attack technique known as Return Oriented Programming (ROP).

CFG works by creating a bitmap of all valid function entry points from within a DLL at compile time. When an indirect call to a function occurs within a module protected by CFG, the bitmap is checked to ensure that the address is that of a valid function entry point. If it is not an exception is thrown. To understand indirection, we can use a simple analogy. If you decide to have pizza for dinner you could go to the store, buy the ingredients, and make it yourself, or, you could order from a pizza delivery restaurant. If you make the pizza yourself, you can feel safe that the pizza has not been contaminated in any way as you are the preparer. If you order the pizza from a restaurant and have it delivered there may be various points of concern. First, you did not witness the making of the pizza, and second, the pizza may have gone through an adventure during delivery to which you are not privy. We do not need to go into the details. This would-be indirection and is heavily based on trust. CFG attempts to help secure this trust.

For more on CFG visit: https://msdn.microsoft.com/en-us/library/windows/desktop/mt637065(v=vs.85).aspx




**More on ROP and CFG**

As stated previously, ROP is a technique often used by attackers to achieve a goal. Often, the goal of ROP is to change the permissions in memory where attacker code resides. We previously discussed DEP and how it marks writable regions of memory as non-executable. By using ROP, one could set up the arguments to a function call such as VirtualProtect() that allows you to change the permissions in memory. ROP works by identifying useful short sequences of code within executable modules and stringing them together to accomplish their goal. These code sequences are referred to as gadgets. We string the gadgets together which formulates our ROP chain. As mentioned, once an attacker gains control of a process, they may wish to change memory permissions so they can have their shellcode executed. To do this a system call must be made to a function like VirtualProtect() or VirtualAlloc(). Control of the process is passed to the gadgets, which returns to each successive gadget, each performing a piece of the overall goal to set up the arguments to the desired function call. CFG limits the number of useful gadgets by only allowing indirect calls to go to addresses indicated in the CFG bitmap.

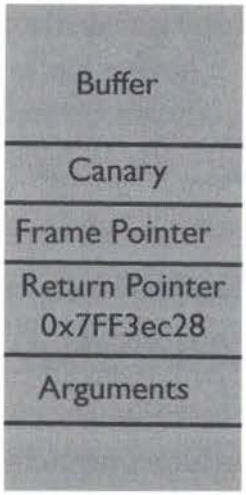
## Stack Canaries / Security Cookies

When a function is called, a return pointer is pushed onto the stack frame for that function.

- The return pointer is used to return control to the caller once the called function is finished
- Overwriting this pointer due to the use of an unsafe function such as strcpy() can result in control hijacking

A canary is a special value placed above the return pointer for protection.

- In order to overwrite the return pointer, the canary must also be overwritten
- If the value is unknown to the attacker then it won't match when it is checked prior to returning control to the caller






**Stack Canaries / Security Cookies**

Stack canaries, also called security cookies in the world of Microsoft, are a compile-time control that inserts code into functions deemed as needing protection. During a normal function call, an address known as the return pointer is pushed into memory onto something known as the procedure stack. Each function call gets its own stack frame on the procedure stack. A stack frame is nothing more than a small amount of memory to store items such as arguments, buffer space, and variables such as the return pointer. Once the function is finished its stack frame is torn down. The return pointer is used to return control to a specific point in the program just after the occurrence of the function call. Since it is stored in writable memory it is prone to being overwritten. If overwritten, control of the process can be hijacked by an attacker. The canary serves as a guard that is pushed onto the stack frame above the return pointer. In order for an attacker to reach the return pointer during an overwrite attempt, they must also overwrite the canary. Most canaries are random and thus an attacker would not know what value to write to that position during an overflow. Prior to returning control to the calling function, the canary is checked to ensure it has not been damaged. If the canary check fails then an exception is thrown and the process terminated.

## Control Flow Integrity (CFI)

Intel released a paper in June 2016 describing new controls to be added:

- Shadow Stacks
- Indirect Branch Tracking

The idea of shadow stacks has been around for well over a decade, such as "Stack Shield" released in 2000: http://www.angelfire.com/sk/stackshield/

Shadow stacks allow only the CALL instruction to push a copy of the return pointer to protected memory.

The return address from the primary stack is checked against the address stored on the shadow stack.

Indirect branch tracking performs edge validation.



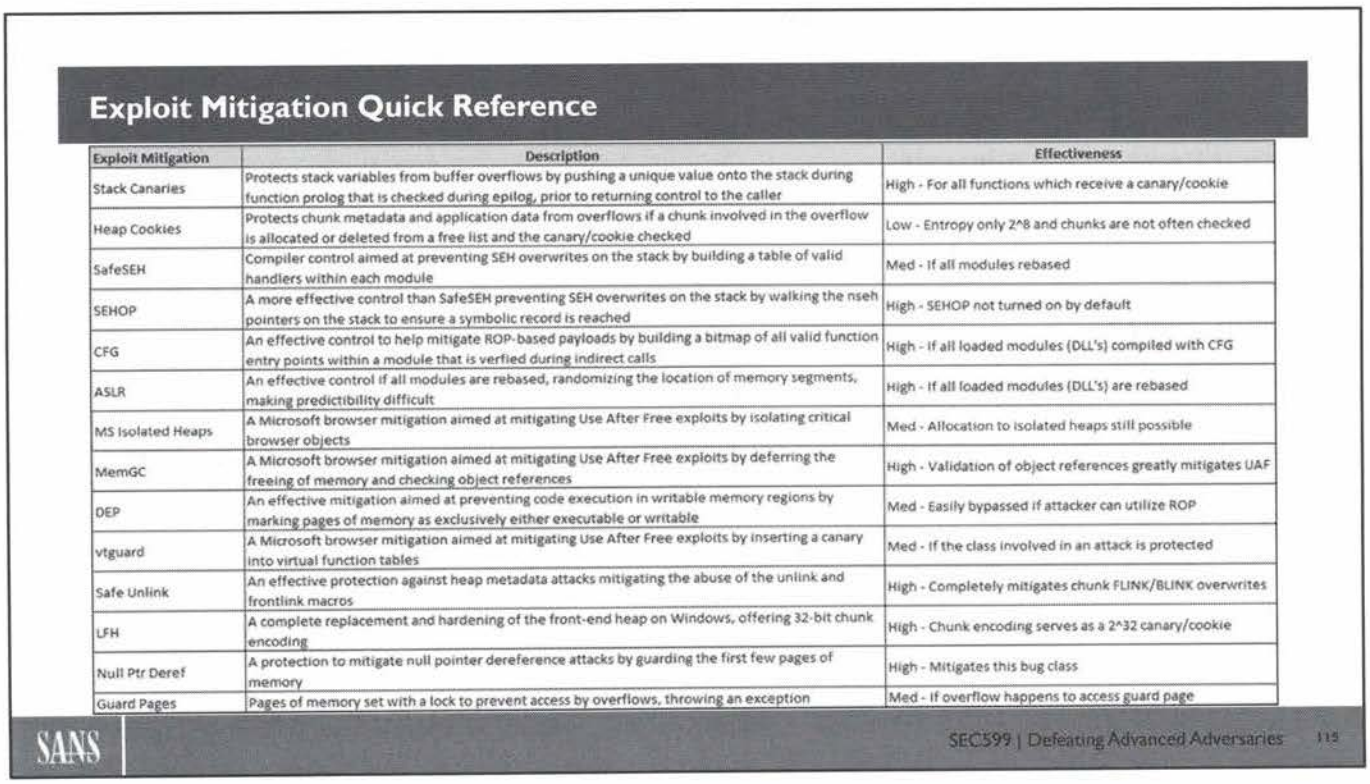
**Control Flow Integrity (CFI)**

In June 2016 Intel released some press releases and a detailed PDF on Control Flow Enforcement (CET), their new upcoming control to prevent code reuse attacks and Return Oriented Programming (ROP) techniques. https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

The primary controls introduced in this paper are Shadow Stacks and Indirect Branch Tracking. Each of these ideas has been proposed in various forms for well over ten years. An example is "Stack Shield" released back in 2000. http://www.angelfire.com/sk/stackshield/ If properly integrated into the processor architecture, each control could have a moderate impact on code reuse techniques. The grsecurity team released a short posting as to their opinion on how Intel's implementation plan of these controls is lacking. https://forums.grsecurity.net/viewtopic.php?f=7&t=4490#P9

Shadow stacks work by marking certain pages of memory as protected, allowing only the CALL instruction the ability to write a copy of the return addresses used in the call chain. The return pointer on the actual stack is tested against the copy stored on the shadow stack. If there is a mismatch an exception is thrown.

Indirect branch tracking takes advantage of the new instruction "ENDBR32" for 32-bit or "ENDBR64" for 64-bit. This instruction is inserted after each valid call instruction. If this is not the next instruction an exception is thrown. The instruction has the same effect as a NOP and simply is used for validation.



## Exploit Mitigation Quick Reference

| Exploit Mitigation | Description | Effectiveness |
|---|---|---|
| Stack Canaries | Protects stack variables from buffer overflows by pushing a unique value onto the stack during function prolog that is checked during epilog, prior to returning control to the caller | High - For all functions which receive a canary/cookie |
| Heap Cookies | Protects chunk metadata and application data from overflows if a chunk involved in the overflow is allocated or deleted from a free list and the canary/cookie checked | Low - Entropy only 2^8 and chunks are not often checked |
| SafeSEH | Compiler control aimed at preventing SEH overwrites on the stack by building a table of valid handlers within each module | Med - If all modules rebased |
| SEHOP | A more effective control than SafeSEH preventing SEH overwrites on the stack by walking the nseh pointers on the stack to ensure a symbolic record is reached | High - SEHOP not turned on by default |
| CFG | An effective control to help mitigate ROP-based payloads by building a bitmap of all valid function entry points within a module that is verfied during indirect calls | High - If all loaded modules (DLL's) compiled with CFG |
| ASLR | An effective control if all modules are rebased, randomizing the location of memory segments, making predictibility difficult | High - If all loaded modules (DLL's) are rebased |
| MS Isolated Heaps | A Microsoft browser mitigation aimed at mitigating Use After Free exploits by isolating critical browser objects | Med - Allocation to isolated heaps still possible |
| MemGC | A Microsoft browser mitigation aimed at mitigating Use After Free exploits by deferring the freeing of memory and checking object references | High - Validation of object references greatly mitigates UAF |
| DEP | An effective mitigation aimed at preventing code execution in writable memory regions by marking pages of memory as exclusively either executable or writable | Med - Easily bypassed if attacker can utilize ROP |
| vtguard | A Microsoft browser mitigation aimed at mitigating Use After Free exploits by inserting a canary into virtual function tables | Med - If the class involved in an attack is protected |
| Safe Unlink | An effective protection against heap metadata attacks mitigating the abuse of the unlink and frontlink macros | High - Completely mitigates chunk FLINK/BLINK overwrites |
| LFH | A complete replacement and hardening of the front-end heap on Windows, offering 32-bit chunk encoding | High - Chunk encoding serves as a 2^32 canary/cookie |
| Null Ptr Deref | A protection to mitigate null pointer dereference attacks by guarding the first few pages of memory | High - Mitigates this bug class |
| Guard Pages | Pages of memory set with a lock to prevent access by overflows, throwing an exception | Med - If overflow happens to access guard page |



**Exploit Mitigation Quick Reference**

We have not covered all of the exploit mitigations shown on this slide, but this can serve as a quick reference to see what each control does from a high level and get an idea as to their effectiveness. The effectiveness is subjective and based on the experience and experiences of each exploit writer. These are the ratings as given by this author.

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

**Securing the Network**
- Network Access Control & 802.1X

**Securing Software**
- Software Development Lifecycle (SDL) & Threat Modeling
- Vulnerability Assessments
- Exercise: Authenticated Scans Using Nessus
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation Using Compile-Time Controls
- Exploit Mitigation Techniques – Exploit Guard, EMET, & Others
- Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**
- OS Hardening & Best Practices
- Endpoint Protection Solutions
- Application Whitelisting to Stop Payload Execution
- Exercise: Configuring AppLocker



This page intentionally left blank.

## Exercise – Exploit Mitigation using Compile-Time Controls

The objective of the exercise is to analyze how exploits can be mitigated using compile-time controls. We will use Visual Studio to compile a vulnerable application with and without compile-time control such as stack canaries.

High-level exercise steps:

1. Compile a program without stack canaries
2. Exploit the program to obtain control and perform exploitation
3. Compile the same program with stack canaries
4. Attempt to exploit the program again, now observing the new behavior



**Exercise – Exploit Mitigation using Compile-Time Controls**
The objective of the exercise is to analyze how exploits can be mitigated using compile-time controls. We will use Visual Studio to compile a vulnerable application with and without compile-time control such as stack canaries.

1. Compile a program without stack canaries
2. Exploit the program to obtain control and perform exploitation
3. Compile the same program with stack canaries
4. Attempt to exploit the program again, now observing the new behavior

For additional guidance & details on the lab, please refer to the LODS workbook.

## Exercise – Exploit Mitigation using Compile-Time Controls – Conclusion

During the exercise, we used compile-time controls to mitigate exploitation, even if a given application is vulnerable.

We will now introduce exploit mitigation software such as EMET, which focuses on stopping exploitation at runtime.

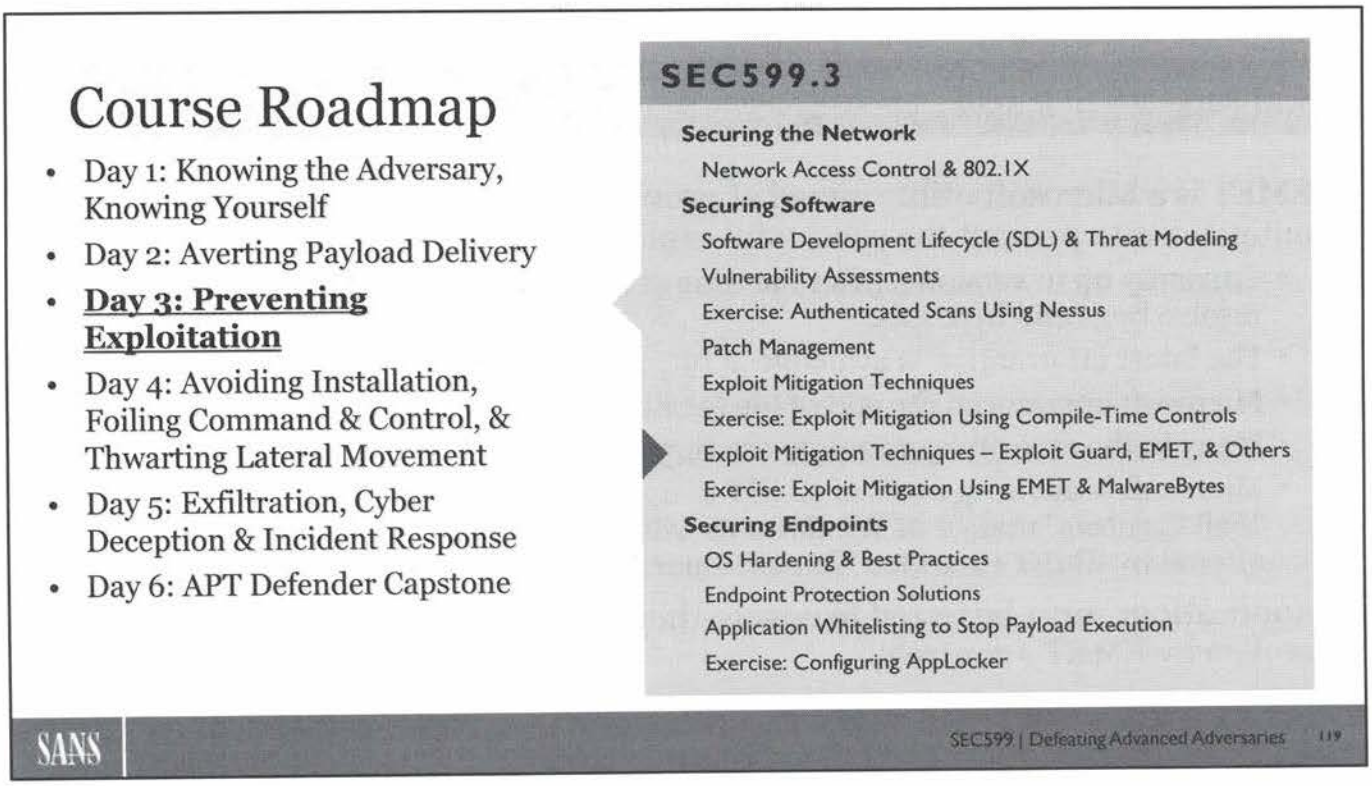
As with many security issues, this is a cat-and-mouse game, and adversaries are increasing their efforts to overcome these exploit mitigation techniques, so we should not only rely on them to prevent exploitation!



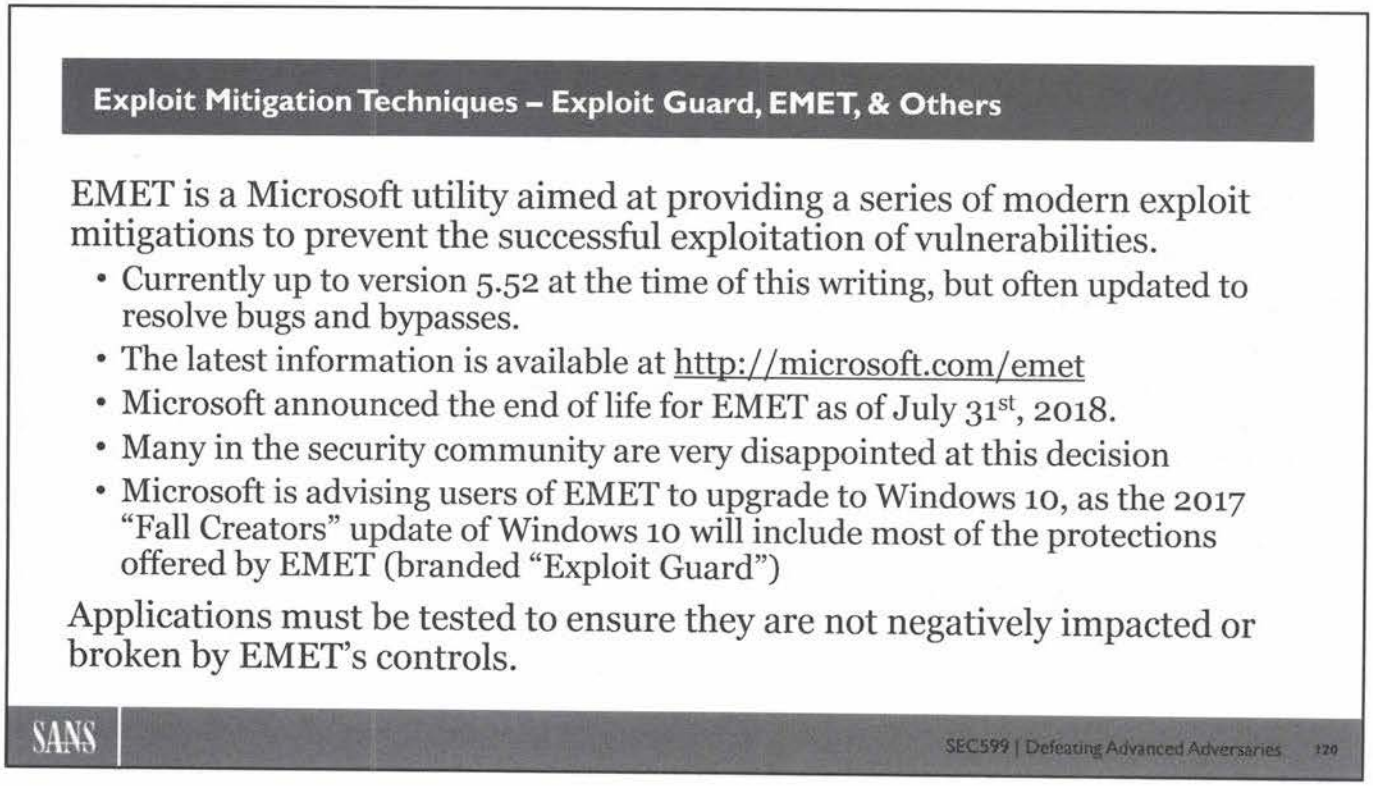
**Exercise – Exploit Mitigation using Compile-Time Controls – Conclusion**
During the exercise, we used compile-time controls to mitigate exploitation when the vulnerable application is being compiled. This is of course only an option if you control the application source code yourself and can compile it. In the next course section, we will introduce exploit mitigation software such as EMET, which focuses on stopping exploitation at runtime.

For us defenders, it's a powerful plus to have an in-depth understanding of how current exploit mitigation techniques work, so we can make the right decisions on what type of protection measures we are to implement in what environment.

As with many security issues, this is a cat-and-mouse game, and adversaries are increasing their efforts to overcome these exploit mitigation techniques, so we should not only rely on them to prevent exploitation!



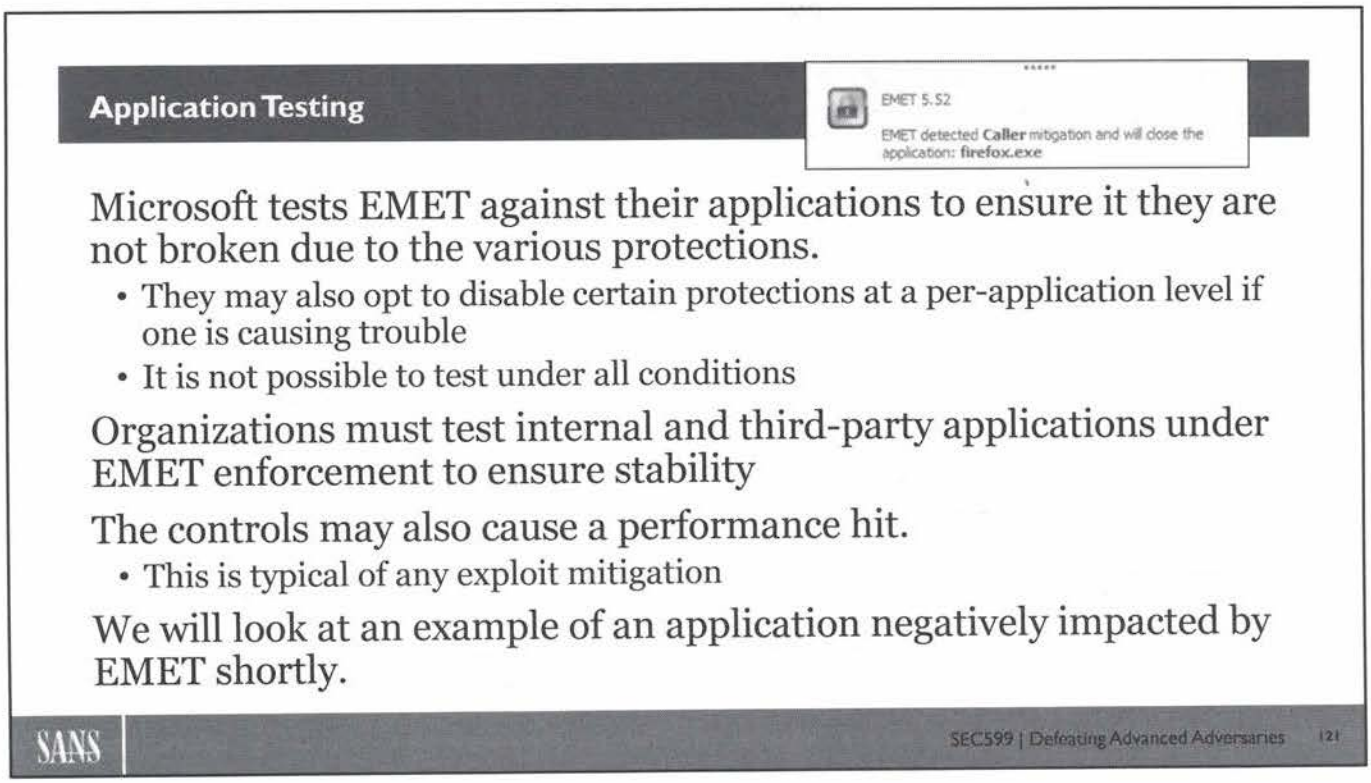

This page intentionally left blank.




**Exploit Mitigation Techniques – Exploit Guard, EMET, & Others**

Microsoft's EMET utility was released back in 2009 around the same time as Windows 7. It offers numerous exploit mitigations aimed at providing defense-in-depth to applications and prevent the successful exploitation of vulnerabilities. At the time of this writing, EMET version 5.52 was the latest available from Microsoft. The latest information and release download is available at http://microsoft.com/emet. EMET releases typically add new protections, general feature improvements, or resolve disclosed bypass techniques. Sadly, Microsoft announced in 2016 that support and development of the product will end on July 31, 2018. Initially, Microsoft meant to discontinue support in January 2017, but due to feedback from customers, they agreed to push back the date. The exact reasoning for the discontinuation of EMET by Microsoft is unclear, though it likely has to do with a low adoption rate over the years and a focus on Windows 10 security and beyond. Many companies are concerned about the negative impact EMET may have on their applications and system performance. EMET was also not initially intuitive to use and lacked good per-application granularity, as well as centralized management which also likely contributed to the low adoption rate.

Microsoft's recommendation as stated at http://microsoft.com/emet is to simply migrate to Windows 10 for improved security. Microsoft is advising users of EMET to upgrade to Windows 10, as the 2017 "Fall Creators" update of Windows 10 will include most of the protections offered by EMET (branded "Exploit Guard").




**Application Testing**

Another point of frustration is around application testing. It requires someone to go through all applications being considered for EMET protection by your organization and check to see if any of the protections cause an issue. It is difficult to say at what point you have done enough testing to deem an application safe to use with EMET. Then, if a new EMET version is released to resolve an EMET bypass technique testing would again be required to ensure the new version doesn't pose any new issues. It is simply not possible to test all scenarios under which an application may run. This is very much similar to how quality assurance (QA) testing occurs during software development. Microsoft EMET comes with a list of default applications that should be protected and generally do not have issues with the utility. These include Microsoft Word, Excel, PowerPoint, Internet Explorer, Edge, Adobe Flash, Java, and several others. If you think about the main applications that fall victim to exploitation at an organization, that list alone should be quite effective.

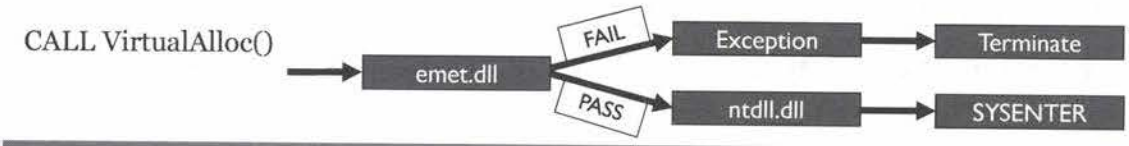
There are also concerns of a system performance hit due to the additional controls. EMET certainly results in extra code execution to perform enforcement. This can slow down an application; however, on a modern workstation, it shouldn't be too big of an issue. Coming up in an exercise, we will take a look at an example of EMET unintentionally breaking an application. In this event, you would simply disable that specific control and document any risk accordingly.

## How Does EMET Work?

The module emet.dll is loaded into all processes designated for protection by EMET.

Many of the controls simply "hook" application flow at specific points.

- An example of hooking is when a table of pointers to various functions is overwritten with pointers to different code
  - This is commonly used by malware, endpoint protection suites, and anti-exploitation products
  - Typically, the originally intended function is reached after going through a series of checks




SANS SEC599 | Defeating Advanced Adversaries 122

**How Does EMET Work?**

A big question is likely, "How does EMET work?" Some of the EMET controls are system-level controls such as DEP, where EMET can control the settings as opposed to going through the system control panel. The more specific per application controls that are native to EMET often work by hooking. This is very similar, if not identical to how many endpoint protection and antivirus products work, as well as malware. Imagine an application wanting to call a function that is deemed critical. Microsoft classifies various functions as critical, such as those with the ability to change permissions in memory, allocate new memory, and many others. When the application goes through the normal channel of calling a critical function the address of that function has been overwritten with an address inside of emet.dll. This allows EMET to perform any checks, and if all looks good control is passed to the desired critical function. We will look at specific examples of controls coming up soon.

## EMET's Graphical Interface

On the right is a screenshot of the EMET GUI from EMET 5.52

In the bottom-right, you can see two processes protected by EMET:
- Microsoft Excel
- Firefox

Windows Explorer is currently not protected.




**EMET's Graphical Interface**

On this slide is a screenshot of EMET 5.52. Some of the buttons and settings are self-explanatory. You can see a mention of controls such as DEP and SEHOP under "System Status." The drop-down box to the right of it would allow you to make any system-wide changes. In the bottom-right, you can see that EMET is protecting Excel and Firefox, but not Windows Explorer. To make changes to any of these settings, you would click on the "Apps" icon on the top ribbon bar. The Certificate Trust (Pinning) control simply allows you to specify Certificate Authorities that are permitted to be used to help prevent man in the middle attacks related to HTTPS.

## EMET Application Configuration (1)

When clicking on the  icon you are presented with the following screen:

Apps




**EMET Application Configuration (1)**

On this slide is a screenshot of the application configuration window. In the bottom-left, you can see the list of applications and to the right are the controls enforced at a per-application level. There are two default actions that can be taken, "Stop on exploit" and "Audit only." Clearly, "Audit only" will only report that a control detected a problem as opposed to throwing an exception and terminating the application. This is useful during application testing to see if EMET is causing any issues.

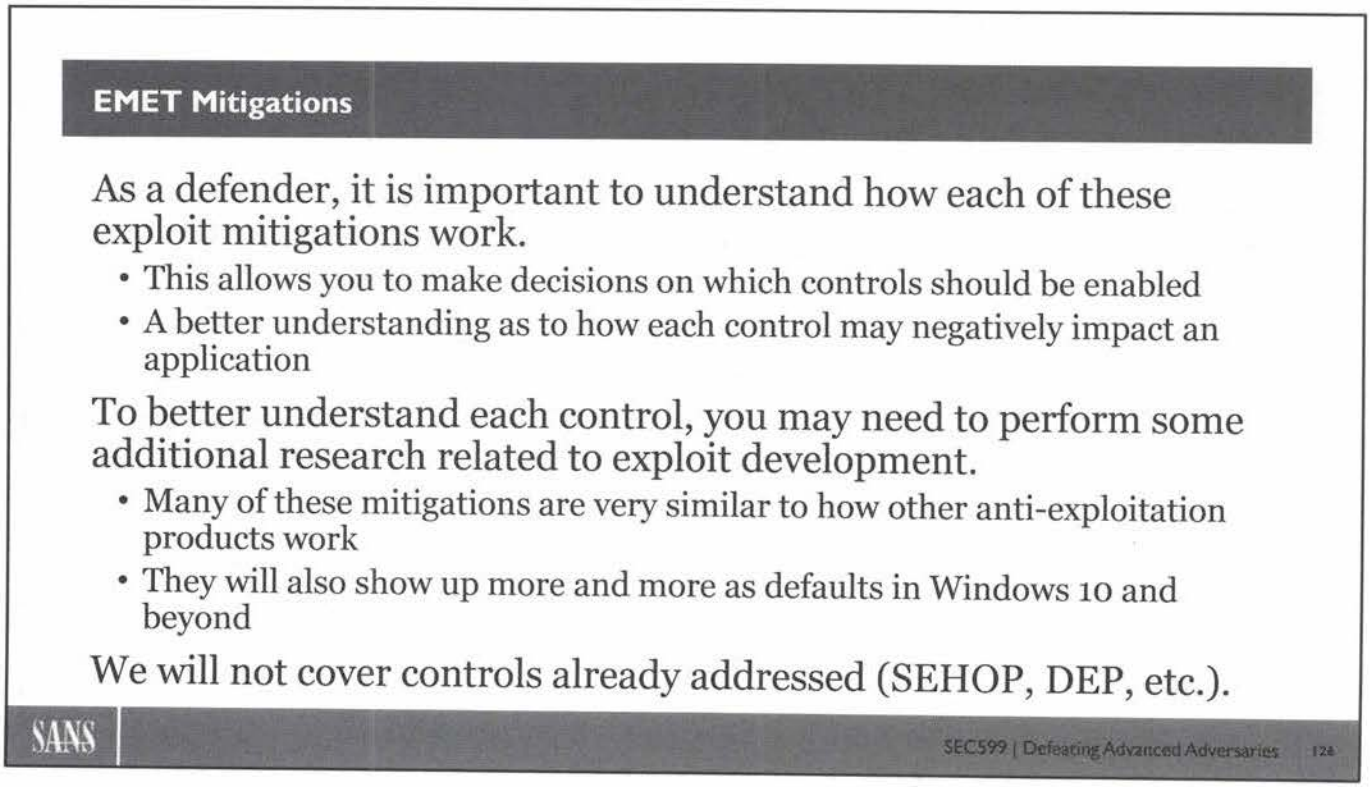
## EMET Application Configuration (2)

When clicking on the "Show All Settings" icon you are presented with detailed information about each control.




**EMET Application Configuration (2)**

As shown on the slide, when clicking on the "Show All Settings" icon detailed information about each control is displayed, as well as the ability to turn them on or off.




**EMET Mitigations**

As a defender, the phrase "Offense must inform the defense" often comes up, but what does that really mean? It is certainly subjective as to how it is to occur. From this author's perspective, to become proficient at a technical area you must put in the time and do the work. You cannot wait for penetration testers, exploit developers, and malware experts to continuously provide information, and even if they could, do you have the prerequisite knowledge to understand what is being said? Take application debugging and reverse engineering as an example. Tools such as the Interactive Disassembler (IDA) and WinDbg are commonly used. They are unintuitive tools each requiring countless hours and practice. An exploit mitigation may be related to a very niche way in how low-level instructions are executed by the processor. Failure to have this prerequisite knowledge can limit your understanding as to the effectiveness. This paragraph is by no means an effort to discourage you. If anything, it should serve as a motivator to say that everyone who is an expert in areas such as malware reversing and exploit development paid their dues. There are few corners available to cut.

Even though EMET may ultimately be retired by Microsoft, many organizations will continue to use the utility. Also, many of the commercial anti-exploitation products out there utilize the same controls. As more and more systems move to Windows 10 it is highly likely that the controls once only available in EMET are moved into the OS as a default. We will not cover controls that we already covered, such as DEP and SEHOP.

## Heap Spray Protection


Heap spraying is a technique commonly used against browsers and other applications to aid in exploitation.

- With controls like ASLR, an attacker may not know if their shellcode is sitting at a specific address
- By making repeated large allocations in memory containing shellcode, eventually, the desired memory address should be reached

The protection works by pre-allocating areas of memory at addresses attackers rely on during a spray.

- The problem is that there may be addresses that aren't on the list

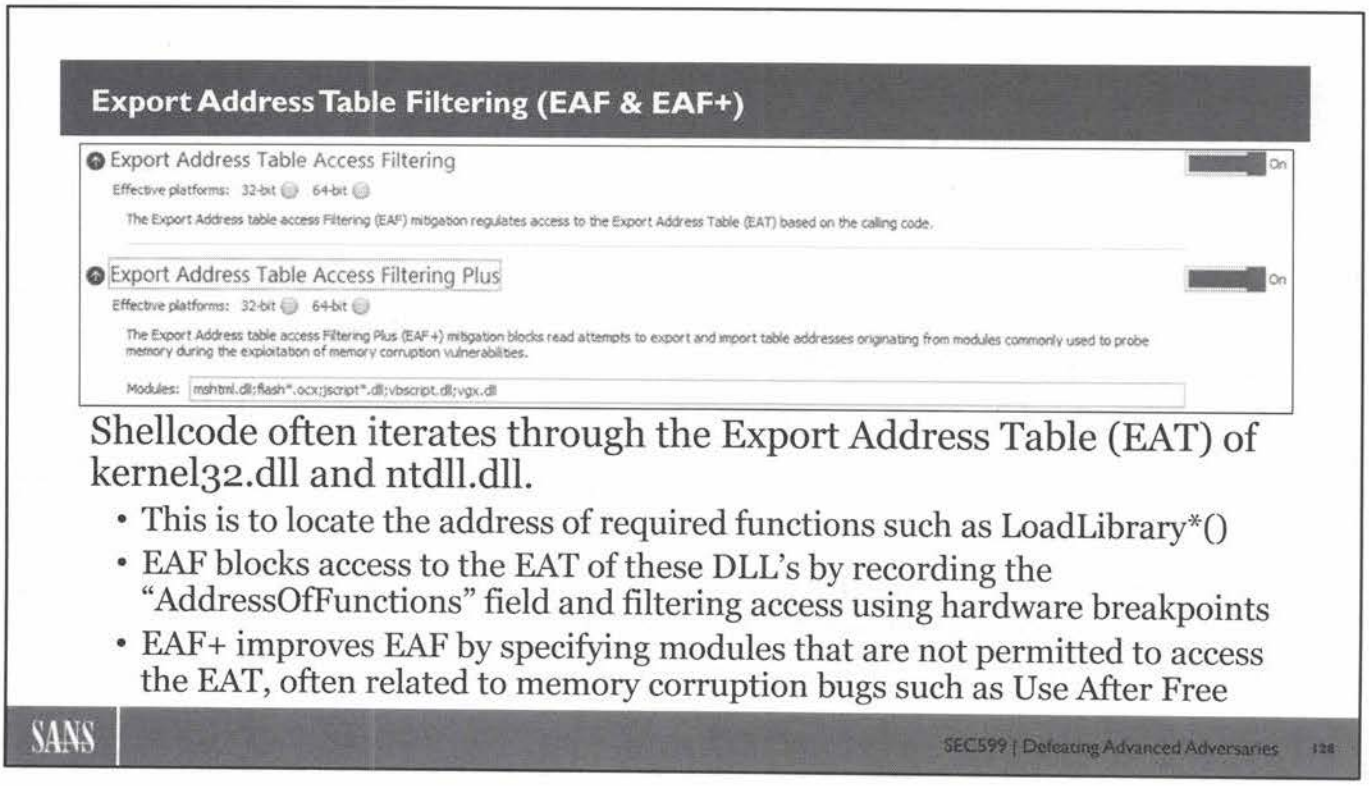


**Heap Spray Protection**

Heap spraying is a technique first made public by the researcher Berend-Jan Wever who goes by the handle "Skylined" as part of his "Internet Exploiter" exploit associated with CVE-2004-1050. The original exploit can be found at https://www.exploit-db.com/exploits/612/. Very little information about how the technique worked was released originally; however, Skylined recently released an article on heap spraying at http://blog.skylined.nl/20161118001.html.

There are a couple of ways that heap spraying can aid during an exploit. One benefit is to help deal with ASLR and the difficulty in knowing if your shellcode will be at a predictable address. Imagine if you were in a small room. You can stand anywhere in the room, but you still only take up the same amount of space. Now, imagine if someone filled ¾ of the room with boxes, pushing you over to the remaining ¼, limiting the space available for you to stand. Now, imagine that it is not you standing there, rather it is shellcode. If we fill up memory by repeatedly making large allocations that contain our shellcode, we will eventually fill up so much memory that we will have extended the region of memory down to a predictable address. Heap spraying also helps attackers during Use After Free (UAF) exploitation.

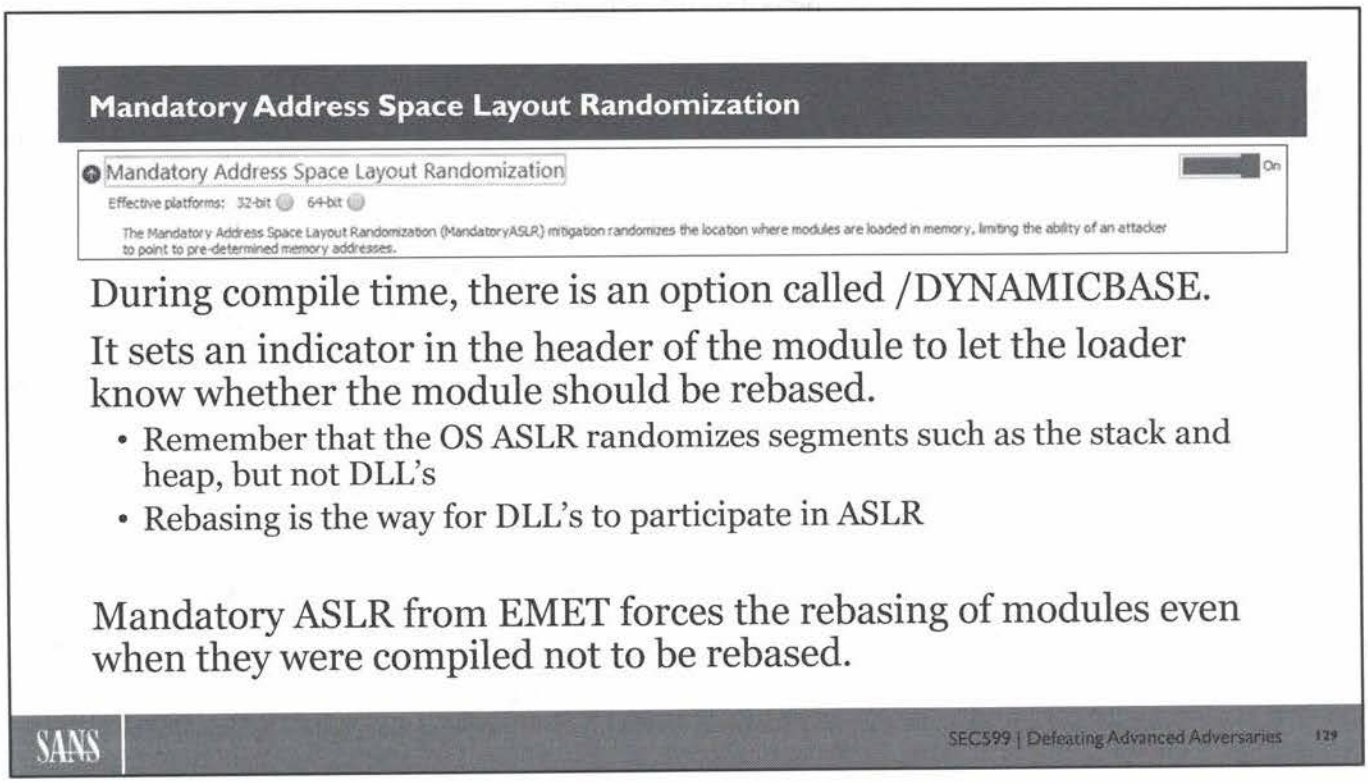
The protection works by pre-allocating the commonly used address that attackers rely on during a heap spray. On the slide, you can see an example of some of these addresses, such as 0x0b0b0b0b and 0x0c0c0c0c. The main issue with the protection is that there are many predictable addresses that can be used, so tracking all of them can be difficult to effectively manage.




**Export Address Table Filtering (EAF & EAF+)**

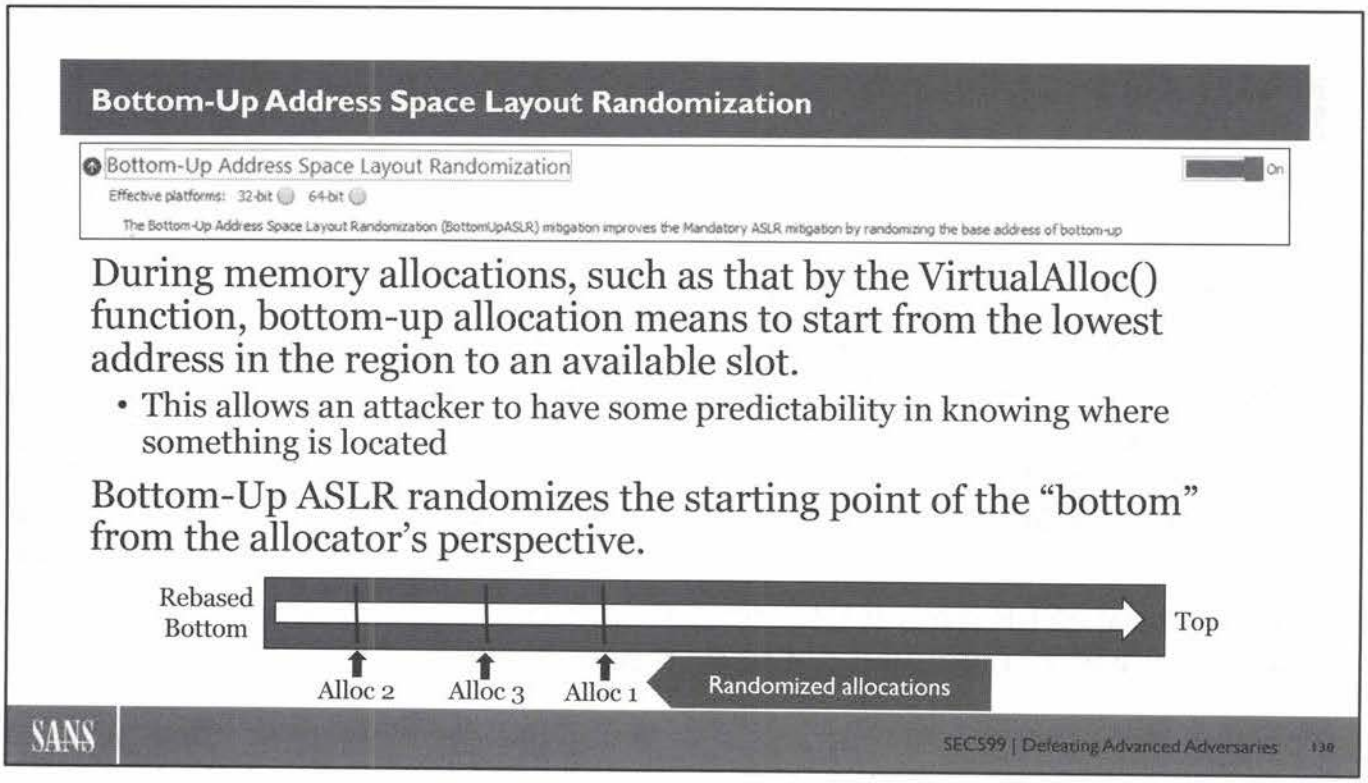
The majority of shellcode for Windows relies on walking through the Export Address Table (EAT) of a DLL in order to resolve the location of its functions. A DLL is a library of functions available for use in applications. An application needs to know where inside the DLL a desired function is located. To make this easy DLL's include an EAT. There is a field called "AddressOfFunctions" which is simply a pointer to an array of pointers, each pointing to the relative virtual address offset of the functions available for use by an application. EAF works by recording the "AddressOfFunctions" field and creating an exception handler. Hardware breakpoints are used when attempting to access the EAT of kernel32.dll and ntdll.dll. The exception handler created by EMET filters access via the hardware breakpoints, breaking access attempts by shellcode. Breakpoints are used by debuggers to pause execution when hitting a specific memory address or under a certain condition. Hardware breakpoints utilize debug registers built into the processor.

EAF+ improves the EAF protection by allowing you to specify modules commonly involved in memory corruption bugs such as Use After Free (UAF) and denying them from reading or writing to export and import address tables of modules such as ntdll.dll, kernel32.dll, and kernelbase.dll. Included by default, as shown above, is mshtml.dll, flash modules, and Visual Basic modules.



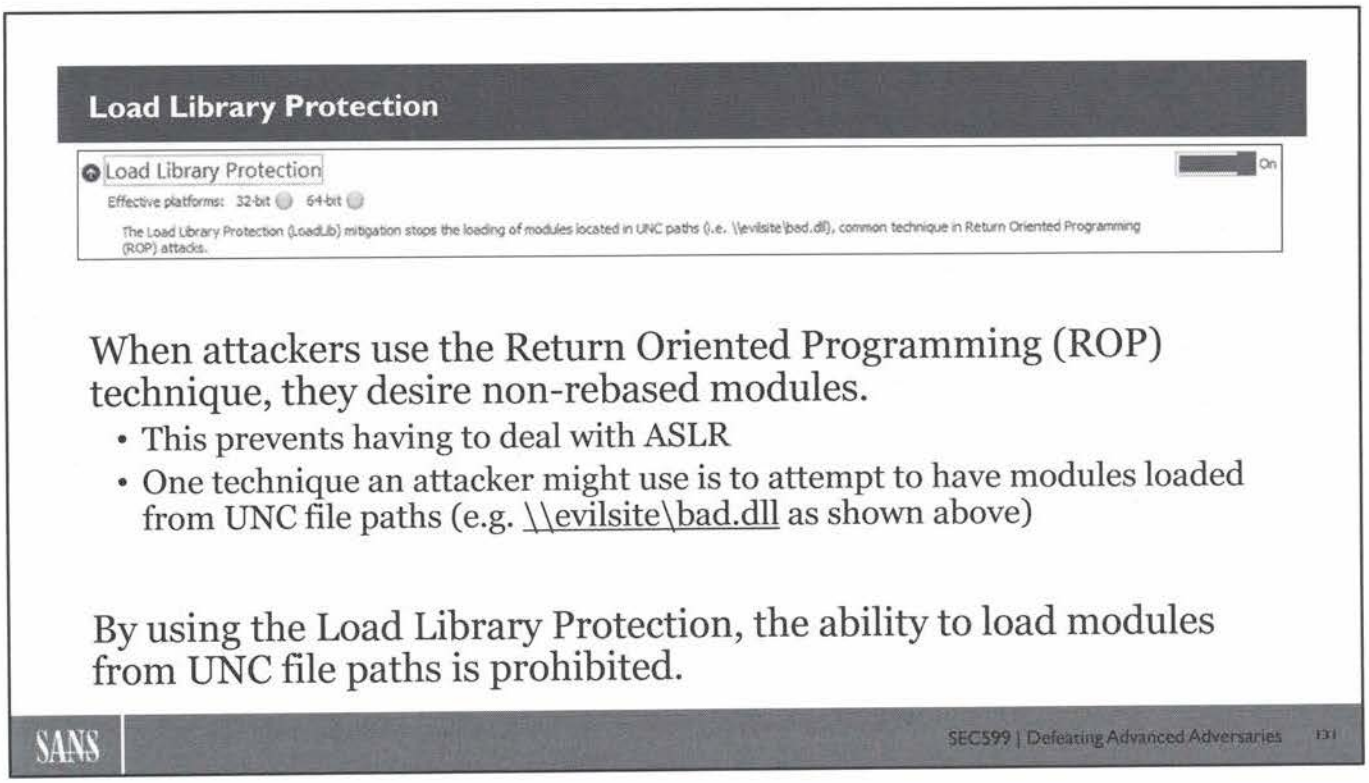

**Mandatory Address Space Layout Randomization**

When compiling a DLL with Visual Studio there is an option called /DYNAMICBASE. Remember, DLL's and modules are the same things, so the words are used interchangeably. When compiled with this option, the header of the DLL is set with an indicator that it is to be rebased when loaded into a process. ASLR, as controlled by the OS, randomizes segments such as the stack and the heap, but DLL's are randomized separately and at a per-DLL level. It is often that an exploit mitigation control can be bypassed due to a single module not participating in a control. Mandatory ASLR from EMET mitigates this issue by forcing the rebasing of all loaded DLL's, regardless of the compiler option set. In theory, this should not cause an issue with an application; however, if there are static addresses used by the application then a crash could occur. The issue of non-rebased modules is typically with the use of 3rd party applications that bring along custom modules to which the application is dependent.



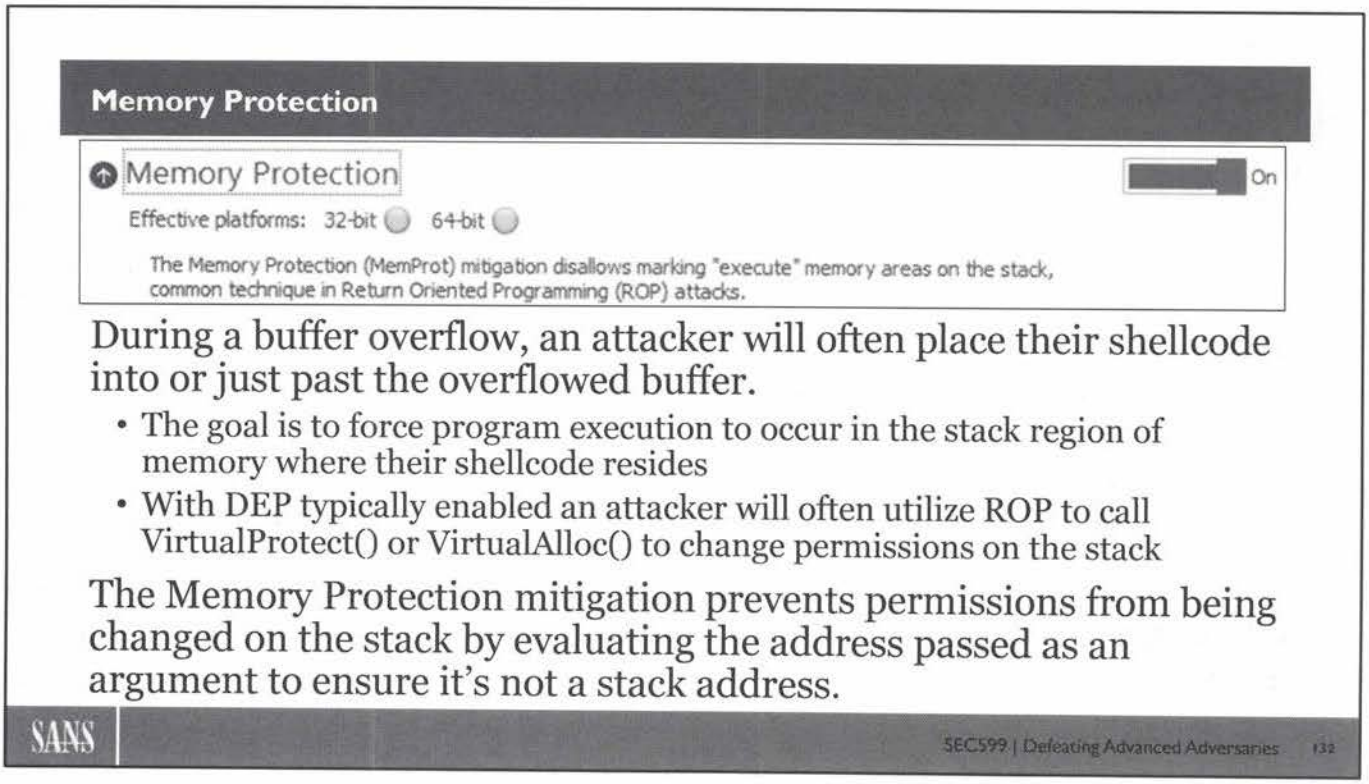

**Bottom-Up Address Space Layout Randomization**

When ASLR randomizes the location of memory segments such as the stack or the heap, a function such as VirtualAlloc() is used to allocate memory in these segments. The allocators typically start from either the top or bottom of the memory segment to find an available slot of memory. Always starting from a location such as the bottom allows for an attacker to have predictability as to where in the segment their data is located. By randomizing all bottom-up allocations this predictability is removed, improving the overall ASLR on the system.



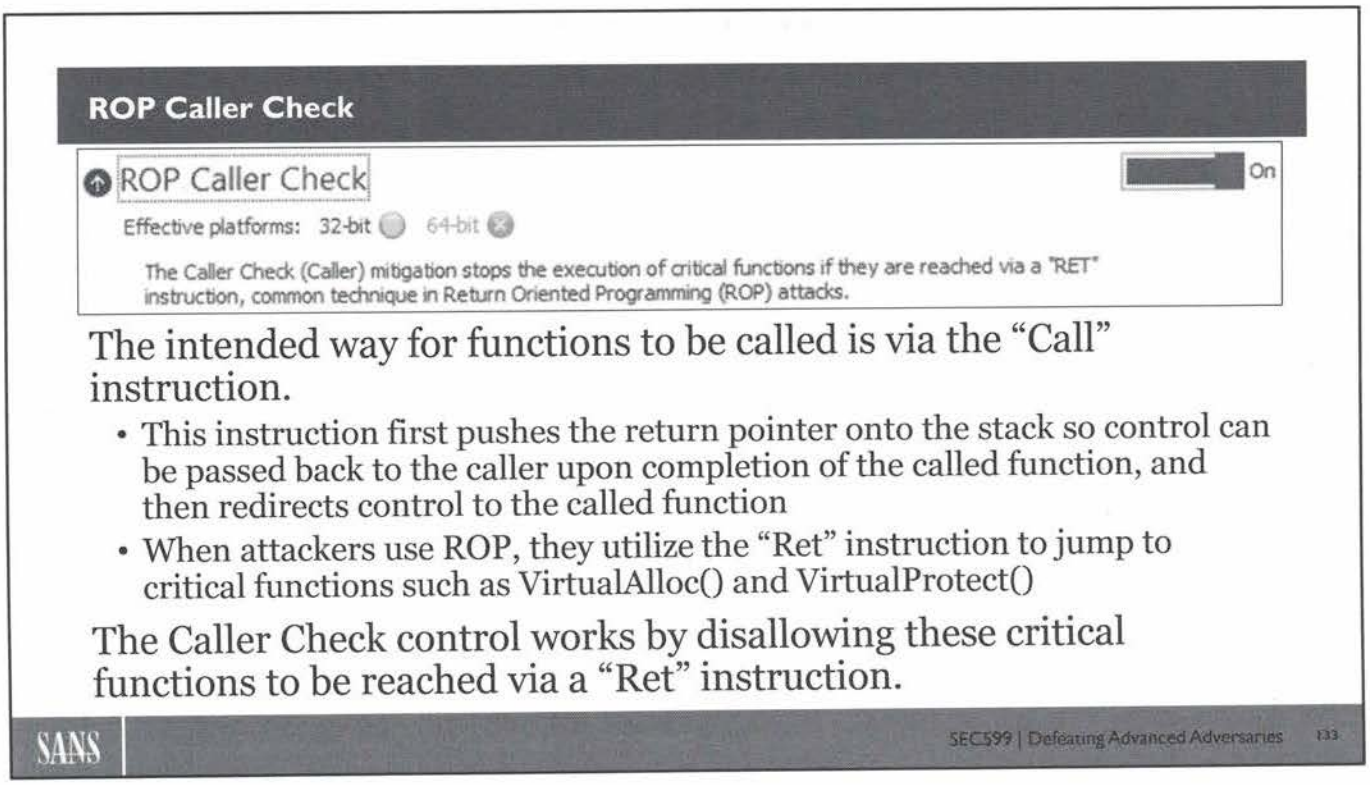

**Load Library Protection**

EMET has multiple controls focused specifically on mitigating Return Oriented Programming (ROP). Load Library Protection is one of these controls. The easiest way for an attacker to create a ROP chain to use in an attack is to have non-rebased modules inside the process from which they can use static addressing. An attacker can attempt to have the application load DLL's from across the network via a UNC file path. This can be leveraged to load modules which contain code desired by the attacker. The Load Library Protection from EMET blocks modules from being loaded via UNC file paths.



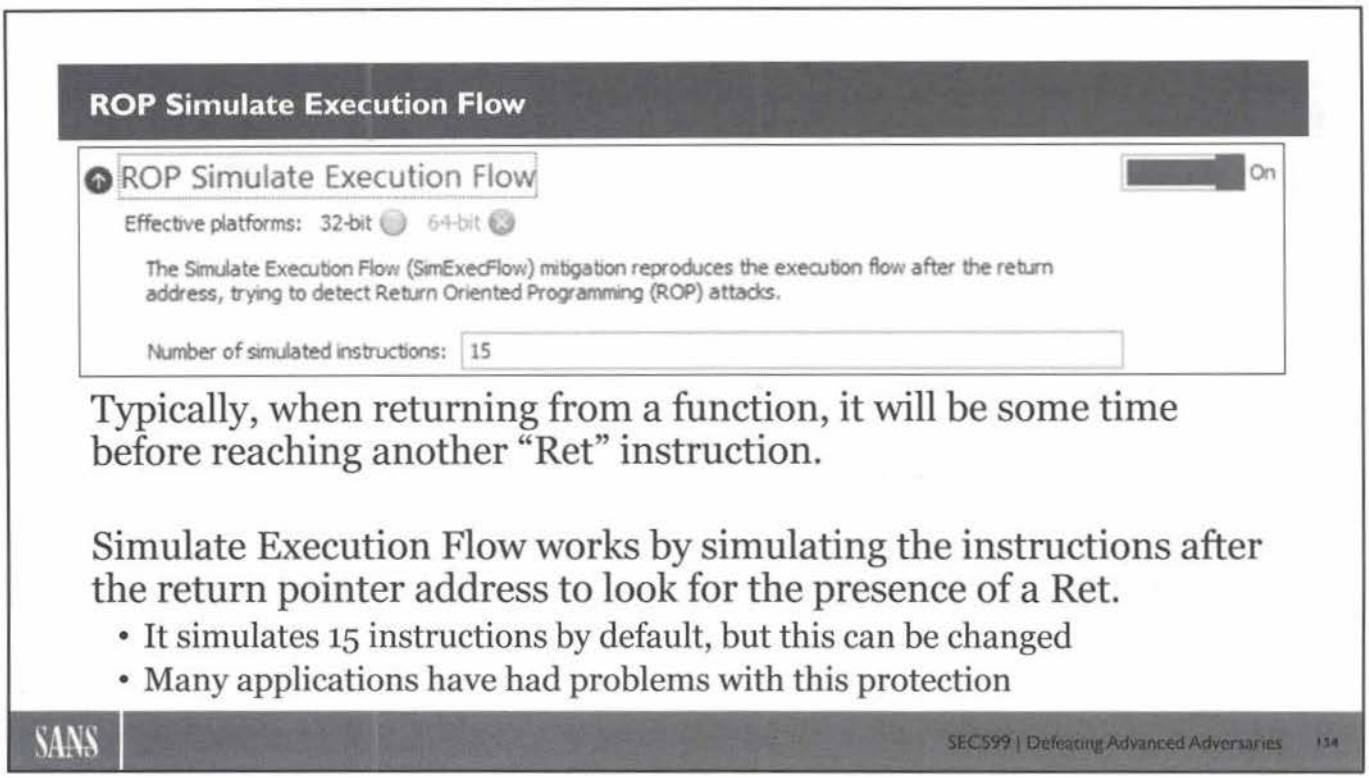

**Memory Protection**

Since the dawn of exploitation, it has been common for an attacker to overflow a buffer on the stack, put their shellcode into or past the overflowed buffer, and return control to this location to execute their payload. It is fairly standard for DEP to be enabled on modern OSes. Attackers typically use Return Oriented Programming (ROP) to call a function such as VirtualAlloc() or VirtualProtect() to change the permissions on the stack where their shellcode is located; otherwise, an exception would be thrown when trying to execute code in a write-only region. The Memory Protection control from EMET works by evaluating the address passed to VirtualProtect(), VirtualAlloc(), or other similar functions to ensure it is not a stack address. It knows what addresses are stack addresses by looking at a structure in memory known as the Thread Information Block (TIB). Each thread within a process gets a unique TIB. One of the items stored in the TIB is the stack limits.



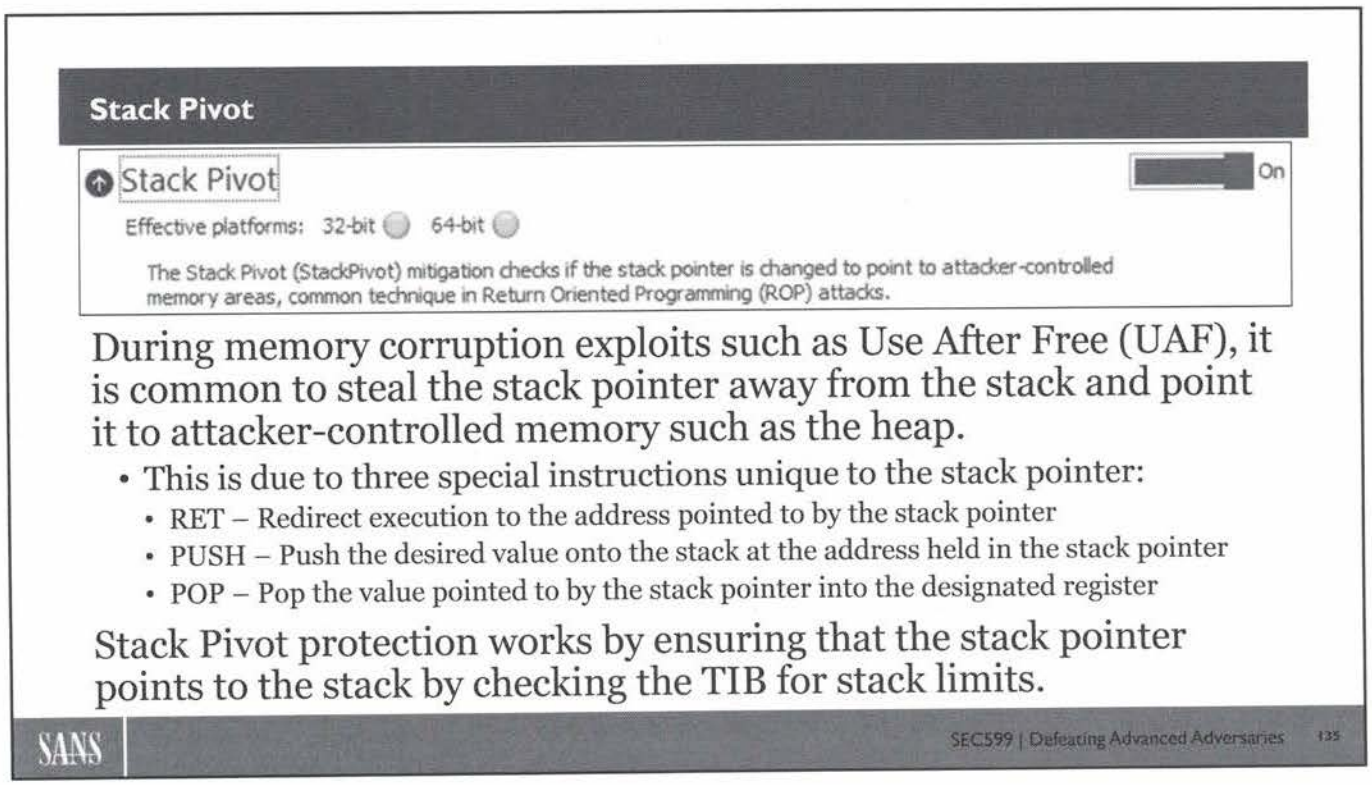

**ROP Caller Check**

In many processor architectures, there is a "Call" instruction. This is the intended way for functions to be called. (e.g. call memcpy) It performs two operations. First, the address of the instruction after the "Call" instruction is pushed onto the stack, serving as the return pointer. This return pointer is used when the called function is finished, allowing for control to be returned to the calling function. The second thing the "Call" instruction does is it redirects control to the actual called function. When attackers utilize ROP as part of their exploit they often rely on the "Ret" instruction to return to the start of critical functions such as VirtualProtect() or VirtualAlloc(). The Caller Check from EMET works by disallowing critical functions from being reached via a "Ret" instruction.




**ROP Simulate Execution Flow**

The Simulate Execution Flow "SimExecFlow" control is like the opposite of the Caller Check. The Call Check makes sure that we are reaching critical functions via a valid "Call" instruction. SimExecFlow works by simulating a predetermined number of instructions (15 is the default) that exist starting with the address to which the return pointer is pointing. It is looking for the existence of instructions commonly used with ROP, most often a "Ret" instruction. When returning from a function call such as VirtualProtect() it is typically a while before you would hit another "Ret" instruction; however, this is not always the case and has been known to cause issues with applications.

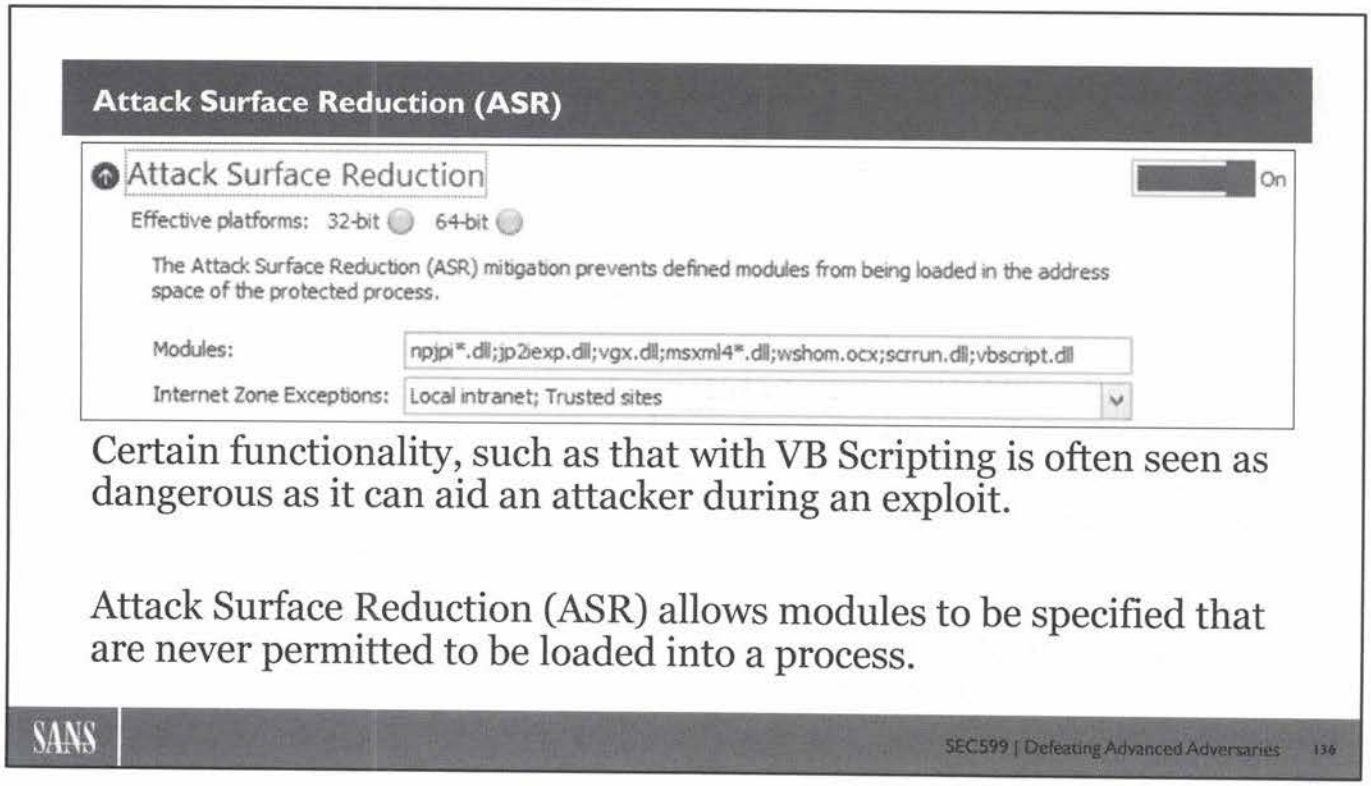



**Stack Pivot**

During memory corruption exploits such as Use After Free (UAF), a common technique is to steal the stack pointer away from pointing to the stack region. This is typically accomplished by using an instruction like "XCHG EAX, ESP". This would cause the EAX register to now point to the stack and the ESP register to point to a region such as the heap. This would be useful if the attacker controls memory on the heap and wishes to leverage unique and powerful instructions like "POP," "PUSH," and "RET" in relation to ROP. Registers are hardcoded variables integrated into the processor cores. They are used for arithmetic operations, storing addresses to memory locations, and many other purposes. Examples of registers include EAX, RIP, CR3, EFLAGS, ESP, R11, etc. The stack pointer is a register (ESP on 32-bit and RSP on 64-bit) that is designed to point to the top of the stack while under the context of a given thread. Those three special instructions are very useful to attackers.

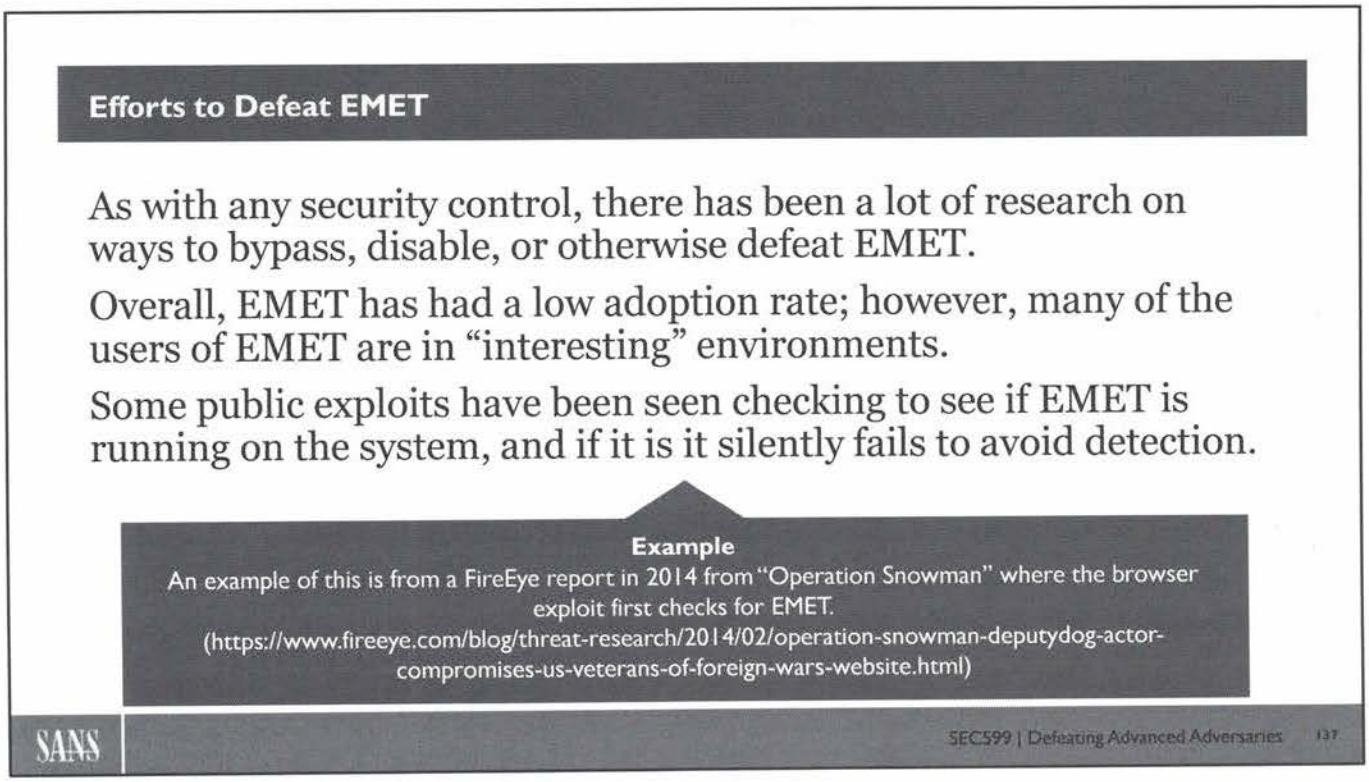
RET – Redirect execution to the address pointed to by the stack pointer
PUSH – Push the desired value onto the stack at the address held in the stack pointer
POP – Pop the value pointed to by the stack pointer into the designated register

The EMET Stack Pivot protection works by checking the stack addressing limits from within the TIB to ensure that the stack pointer is pointing to a stack location.




**Attack Surface Reduction (ASR)**

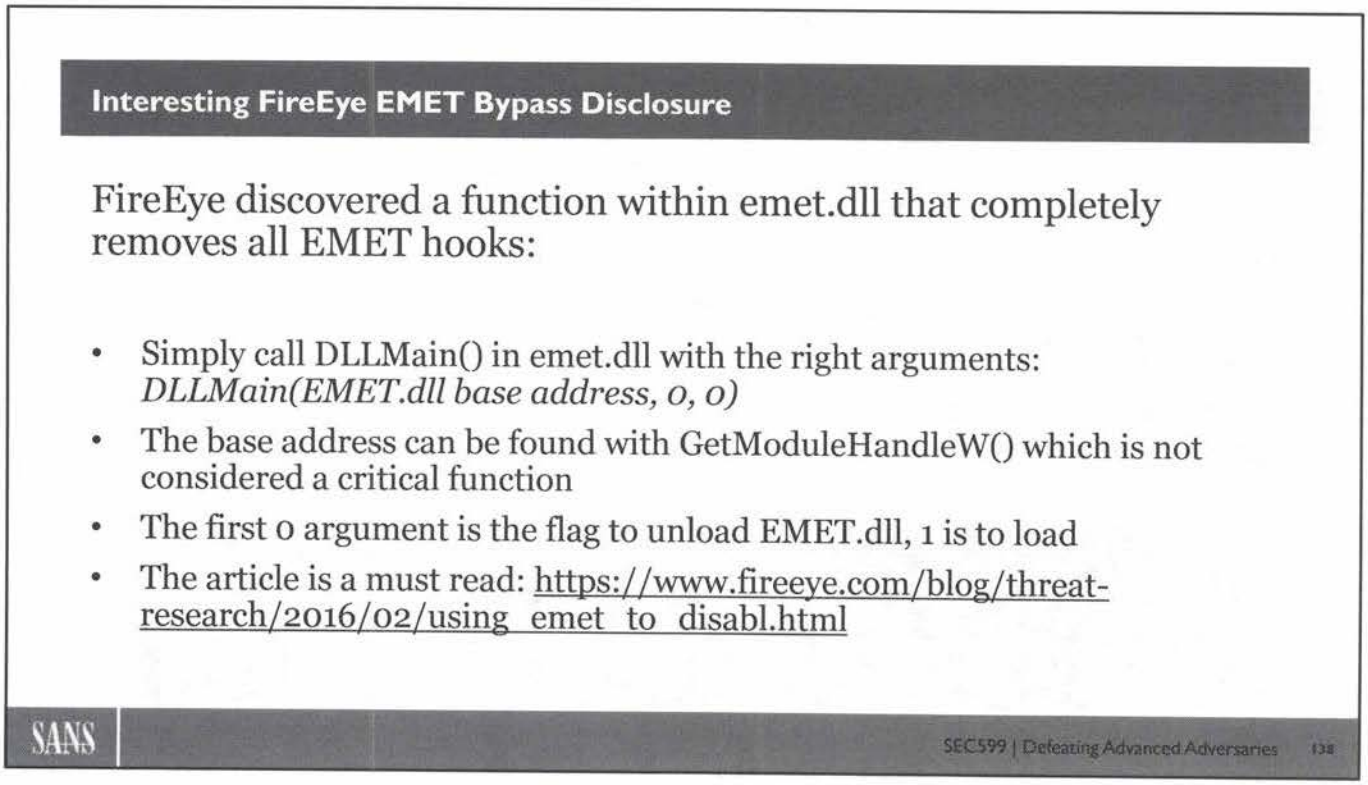
There are quite a few modules that have been involved in many exploits over the years due to the functionality they provide. A couple examples include vgx.dll (Vector Markup Language support), vbscript.dll (Visual Basic Scripting support), and jp2iexp.dll (Java plug-in). Attack Surface Reduction (ASR) allows you to specify any DLL you wish to never be loaded into a process.




**Efforts to Defeat EMET**

Every time a new security feature or device is introduced into the wild, researchers, attackers, and others look for ways to defeat its controls. This is actually a good thing from a security perspective as too much trust has been given to vendors of security products, such as antivirus software. EMET has had a relatively low adoption rate over the years due to numerous reasons. This is likely part of the reasoning for Microsoft's discontinuation of EMET in 2018. Many of the users and organizations using EMET are from interesting lines of work, including government, defense, critical infrastructure and others. This would also add to the draw of finding ways around the tool.

FireEye released a paperback in 2014 showing a browser exploit that first checked the victim to see if they had EMET running. If so, it silently fails to avoid detection. This was clearly an effort to keep the exploit unknown for as long as possible. You can check out the details here: https://www.fireeye.com/blog/threat-research/2014/02/operation-snowman-deputydog-actor-compromises-us-veterans-of-foreign-wars-website.html




**Interesting FireEye EMET Bypass Disclosure**

FireEye also released an interesting article on a bypass they discovered. To summarize, once you gain control of the process you need to deal with EMET prior to attempting the disabling of DEP and execution of your shellcode. Much of the research has been quite complex in relation to methods to bypass EMET; however, this technique simply requires that you locate the base address of emet.dll and replay the DLLMain() function with an argument of 0 to unload all hooks. Pretty amazing.

References:

Alsaheel, Abdulellah. Pande, Raghav. "Using EMET to Disable EMET." FireEye Using EMET to Disable EMET. https://www.fireeye.com/blog/threat-research/2016/02/using_emet_to_disabl.html (accessed February 1, 2017).

## EMET Bypass – Additional Resources

Some additional resources that can prove to be useful for bypassing EMET include:

- https://duo.com/assets/pdf/wow-64-and-so-can-you.pdf
  by Darren Kemp & Mikhail Davidov
- http://labs.bromium.com/2014/02/24/bypassing-emet-4-1/
  by Jared DeMott
- http://casual-scrutiny.blogspot.com/2016/02/cve-2015-2545-itw-emet-evasion.html
  by r41p41
- https://www.offensive-security.com/vulndev/disarming-and-bypassing-emet-5-1/
  by Offensive Security
- http://0xdabbad00.com/wp-content/uploads/2013/11/emet_4_1_uncovered.pdf
  by Dabbadoo



**EMET Bypass – Additional Resources**
Some additional resources that can prove to be useful for bypassing EMET include:

https://duo.com/assets/pdf/wow-64-and-so-can-you.pdf
by Darren Kemp & Mikhail Davidov

http://labs.bromium.com/2014/02/24/bypassing-emet-4-1/
by Jared DeMott

http://casual-scrutiny.blogspot.com/2016/02/cve-2015-2545-itw-emet-evasion.html
by r41p41

https://www.offensive-security.com/vulndev/disarming-and-bypassing-emet-5-1/
by Offensive Security

http://0xdabbad00.com/wp-content/uploads/2013/11/emet_4_1_uncovered.pdf
by Dabbadoo

## Malwarebytes

Malwarebytes is a commercial anti-malware product for Windows, Mac OS, and Android devices.

- A free version is offered with limited functionality, as well as a trial version

Protections are offered against malware, exploitation techniques, and ransomware.

For the purpose of this module, we are looking at the anti-exploit portion of the product.

- Similar to Microsoft's EMET and an alternative as EMET is EOL in 2018.
- Focuses on the most commonly exploited applications such as IE, Chrome, Office, Flash, etc.



**Malwarebytes**

Malwarebytes Anti-Exploit (MBAE) is a commercial alternative to Microsoft's Enhanced Mitigation Experience Toolkit (EMET), which as stated previously, is set to be end of life in mid-2018. Even if all systems were running Windows 10, there are still many EMET-provided exploit mitigations not supported natively. This justifies the case to consider alternative products. Malwarebytes has been around for over ten years initially offering only anti-malware type features, such as the removal of Spyware and Adware, as well as identifying common infections through scanning. As the product evolved real-time scanning was incorporated, as well as anti-exploitation functionality like EMET and Ransomware protection. A free version is available once the 14-day trial expires offering anti-malware and anti-spyware protection, as well as rootkit detection, as stated on their website at https://www.malwarebytes.com/mwb-download/.

For the purposes of this module, our attention is focused on the anti-exploitation functionality. To try and simplify configuration and focus on the most common targets involved in exploitation, MBAE focuses on browsers, Flash, MS Office Suite, PDF readers, and media players.

References:

Malwarebytes. "Malwarebytes Endpoint Security." MBAEBGuide.pdf https://www.malwarebytes.com/pdf/guides/MBAEBGuide.pdf (January 26th, 2017).

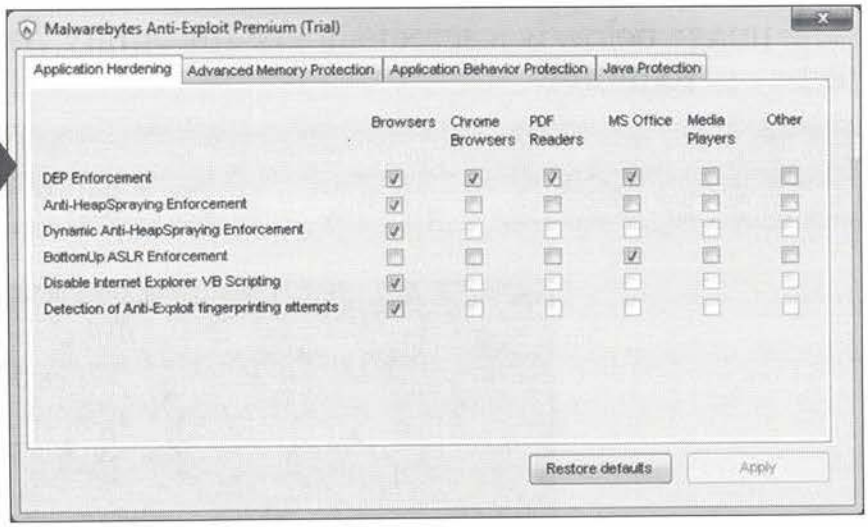
## Malwarebytes Anti-Exploit (MBAE)

**MBAE GUI**

Under the advanced settings menu for MBAE, you can see the different categories of protections:

- Application Hardening
- Advanced Memory Protection
- Application Behavior Protection
- Java Protection

Many of the protections are similar to EMET with some additional ones too…






**Malwarebytes Anti-Exploit (MBAE)**

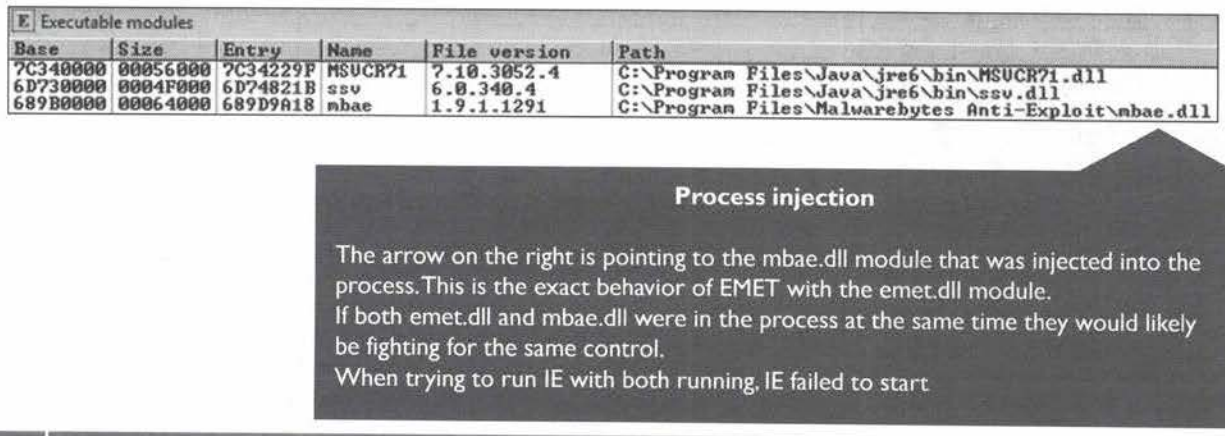
When looking at the trial version of MBAE, you can go to the "Advanced settings" menu from the control panel to bring up the image on the slide. The tabs at the top include "Application Hardening," "Advanced Memory Protection," Application Behavior Protection," and "Java Protection." Many of the controls should look familiar as they are similar to EMET, including "DEP Enforcement," "Anti-HeapSpraying Enforcement," "BottomUp ASLR Enforcement" and many others, some not offered by EMET. The techniques behind the controls are very similar, if not identical to EMET. MBAE and EMET should not be running on the same system as both will inject a DLL into the applications chosen for protection when started and attempt to apply the same types of hooks. This will likely end badly.

MBAE has additional controls over EMET focusing on Macro abuse of WMI and Visual Basic for Applications (VBA), as well as Java protections focused on common payloads such as Meterpreter.

Check out the following reference for more information:
https://www.malwarebytes.com/pdf/guides/MBAEBGuide.pdf

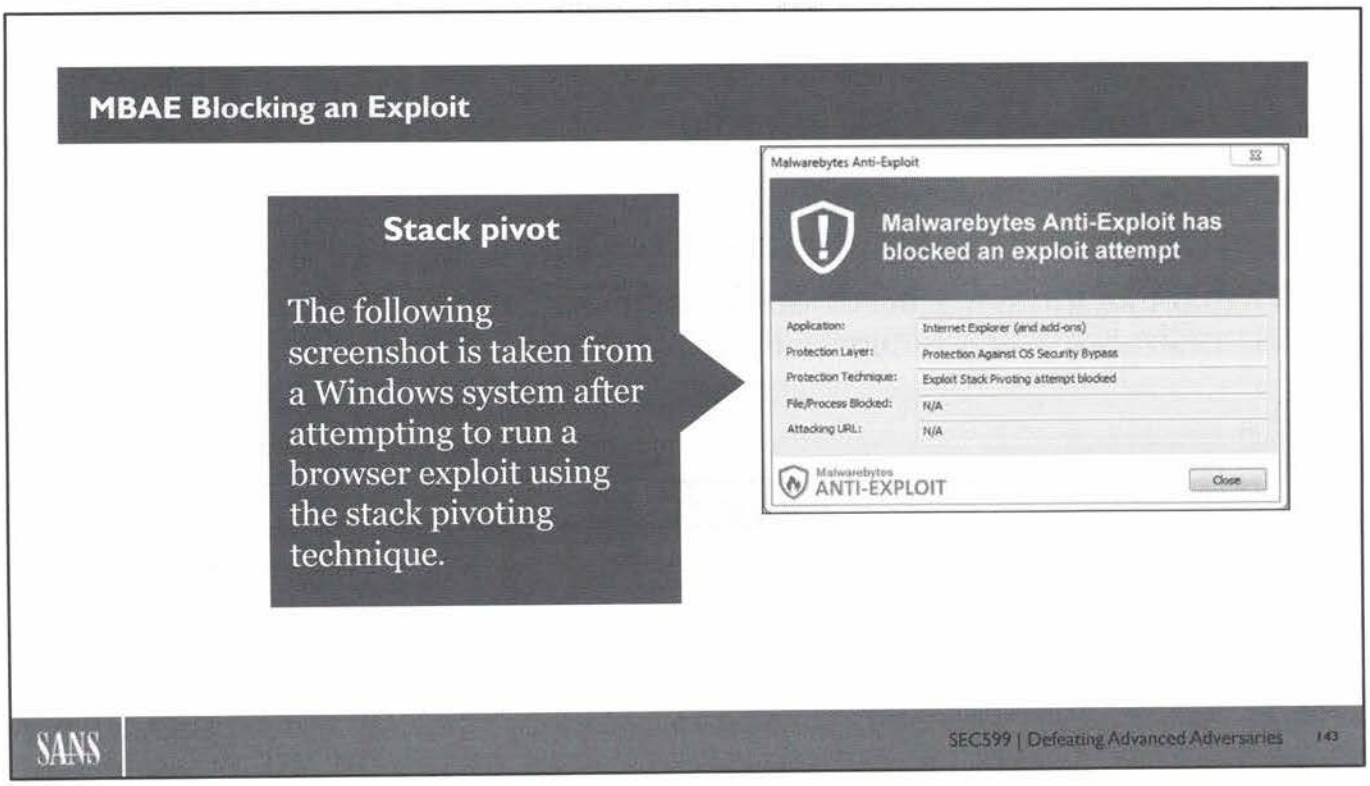
## Looking at MBAE with Immunity Debugger

The image below is a screenshot of Immunity Debugger attached to Internet Explorer:



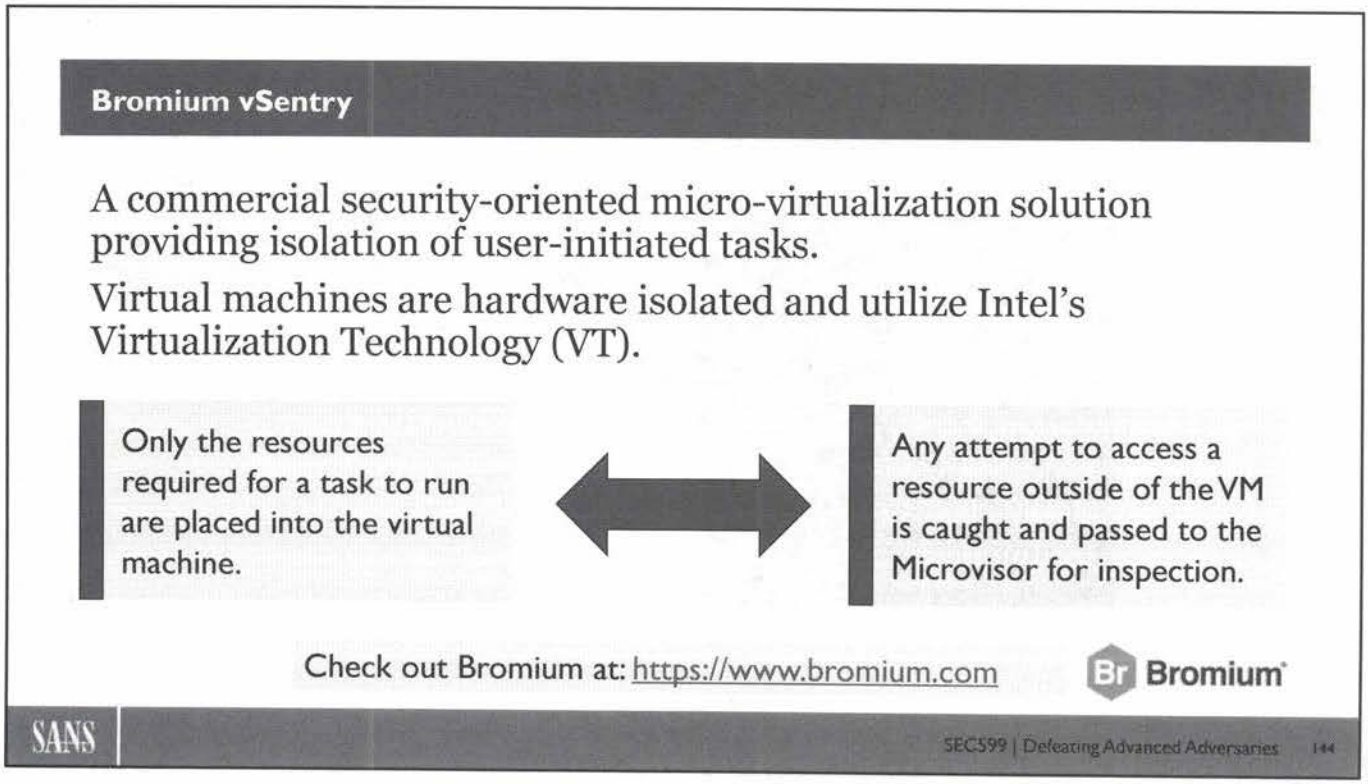

**Looking at MBAE with Immunity Debugger**

On this slide is a screenshot from an Immunity Debugger session attached to Internet Explorer. MBAE is installed on the system and you can see that the module mbae.dll is injected into the process as indicated by the arrow. This behavior is identical to EMET. If both emet.dll and mbae.dll were loaded into this process at the same time the would likely be fighting for the same control. As a test to see what would happen this author ran EMET, protecting Internet Explorer, and also MBAE. The process failed to start after several attempts with no alerts from EMET or MBAE and no logs shown in Windows Event Viewer.




**MBAE Blocking an Exploit**

This slide simply shows the alert box that appeared after running a browser exploit on a system protected by MBAE that uses the stack pivoting technique covered earlier.

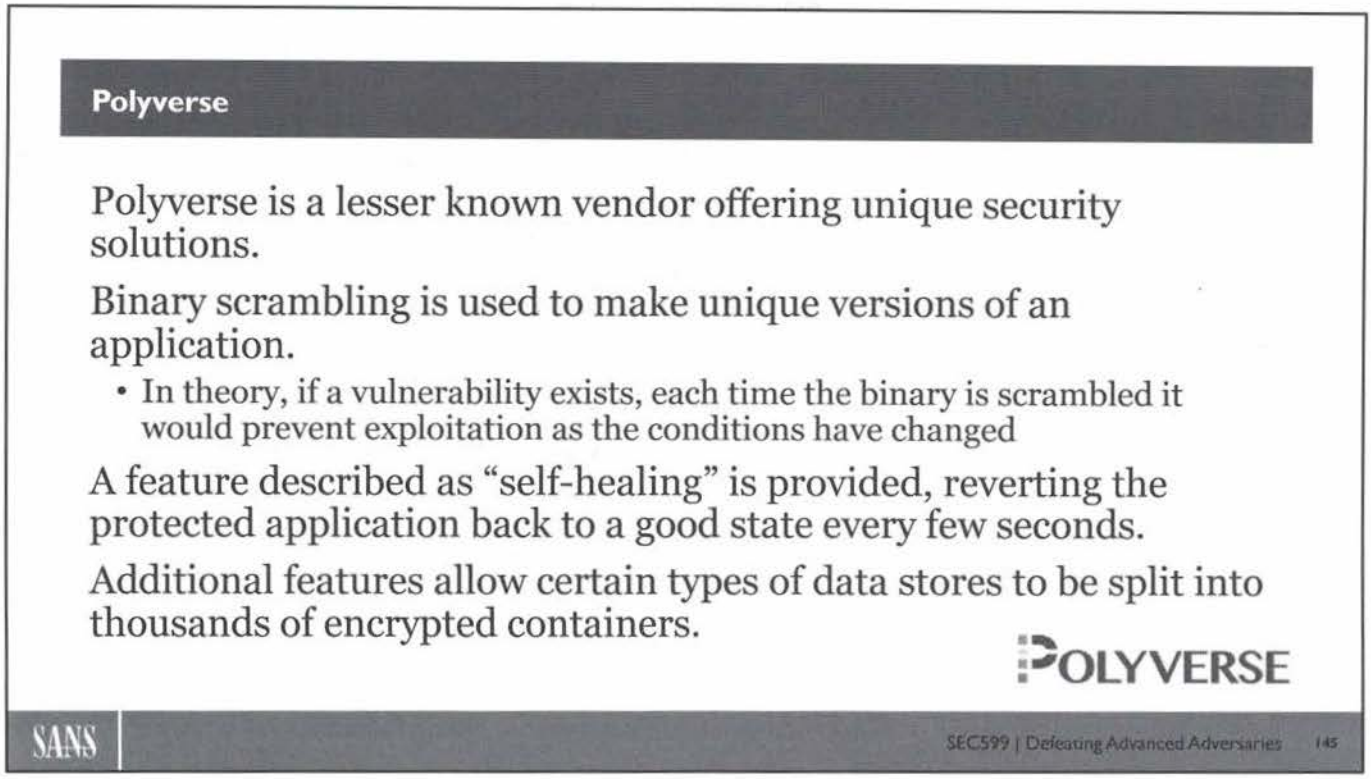



**Bromium vSentry**

A couple of commercial security solutions offer micro-virtualization in where portions of the operating system or processes are contained within their own virtual machine, preventing wide-scale system access. Bromium is a vendor offering a product called vSentry which isolates user initiated tasks using Intel's Virtualization Technology (VT). Each task is provided with only the resources necessary to properly run. When a task attempts to interact with another task or anywhere outside of its own VM, hardware interrupts occur and the request is passed to the Microvisor for inspection and application of a set of mandatory access controls.

References:
Bromium. "vSentry." vSentry | Bromium. https://www.bromium.com/node/147.html (accessed January 26th, 2017).

Bromium copyrighted image taken from: https://www.bromium.com/




**Polyverse**

Another contender in the space of vendors such as Bromium is Polyverse. There is not too much publicly available information about the internal technology of their product; however, unique features include the concept of binary scrambling and "self-healing." You may be familiar with the idea of polymorphic malware. This is malware that attempts to evade detection by constantly changing so that signatures go unmatched. Binary scrambling takes advantage in a similar fashion in that the binary is changed from its original state. If you are familiar with assembly code you know that there are many ways for instructions to achieve the desired result.

Let's say we want the x64 RAX processor register to hold a value of 0x40. In assembly, we have several ways to accomplish this goal. The following is a simple example:

```
e.g. 1
xor rax, rax                    # Zero out the RAX register
mov al, 0x40                    # Move 0x40 into the lower byte of the RAX register (al stand for accumulator low)

e.g. 2
mov rax, 0xffffffc0             # Move 0xffffffc0 into the RAX register
neg rax                         # Compute the two's complement of the value stored in the RAX register, resulting in
0x40
```

Another feature offered by Polyverse is what they describe as "self-healing." This is simply the application reverting back to a known good state every few minutes. Additional advanced features offer the splitting of a data store such as a database into thousands of encrypted containers. This is all definitely a technology to keep an eye on.

References:

Polyverse. "Disrupting hackonomics." Polyverse Technology. https://polyverse.io/technology/ (January 26th, 2017).

Polyverse copyrighted image taken from https://polyverse.io/

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

**Securing the Network**
- Network Access Control & 802.1X

**Securing Software**
- Software Development Lifecycle (SDL) & Threat Modeling
- Vulnerability Assessments
- Exercise: Authenticated Scans Using Nessus
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation Using Compile-Time Controls
- Exploit Mitigation Techniques – Exploit Guard, EMET, & Others
- Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**
- OS Hardening & Best Practices
- Endpoint Protection Solutions
- Application Whitelisting to Stop Payload Execution
- Exercise: Configuring AppLocker



This page intentionally left blank.

## Exercise – Exploit Mitigation using EMET & MalwareBytes

The objective of the exercise is to analyze how exploits can be mitigated in using EMET. We will install a vulnerable application (IceCast) that can be exploited by publicly available tools.

High-level exercise steps:

1. Install a vulnerable application (IceCast)
2. Exploit the vulnerable application using Metasploit
3. Install & configure EMET
4. Re-attempt exploitation



**Exercise – Exploit Mitigation using EMET & MalwareBytes**

The objective of the exercise is to analyze how exploits can be mitigated in using EMET & MalwareBytes. We will also assess the risks of using EMET, as we will see how it can break an unsupported application.

The high-level steps of the exercise include:

- As a first step, we will install a vulnerable software called "Icecast" to demonstrate an exploitable piece of software;
- We will exploit the vulnerable software using Metasploit;
- We will then install & configure EMET to protect our system;
- We will re-attempt exploitation, thereby illustrating how the attack is now blocked;

## Exercise – Hands-on Exploit Mitigation – Conclusion

During the exercise, we used different ways of mitigating exploitation, even if a given application is vulnerable. Depending on your environment, any of these techniques could be beneficial, while a lot of them could already be implemented in your base OS.

As with many security issues, this is a cat-and-mouse game, and adversaries are increasing their efforts to overcome these exploit mitigation techniques, so we should not only rely on them to prevent exploitation!


Malwarebytes

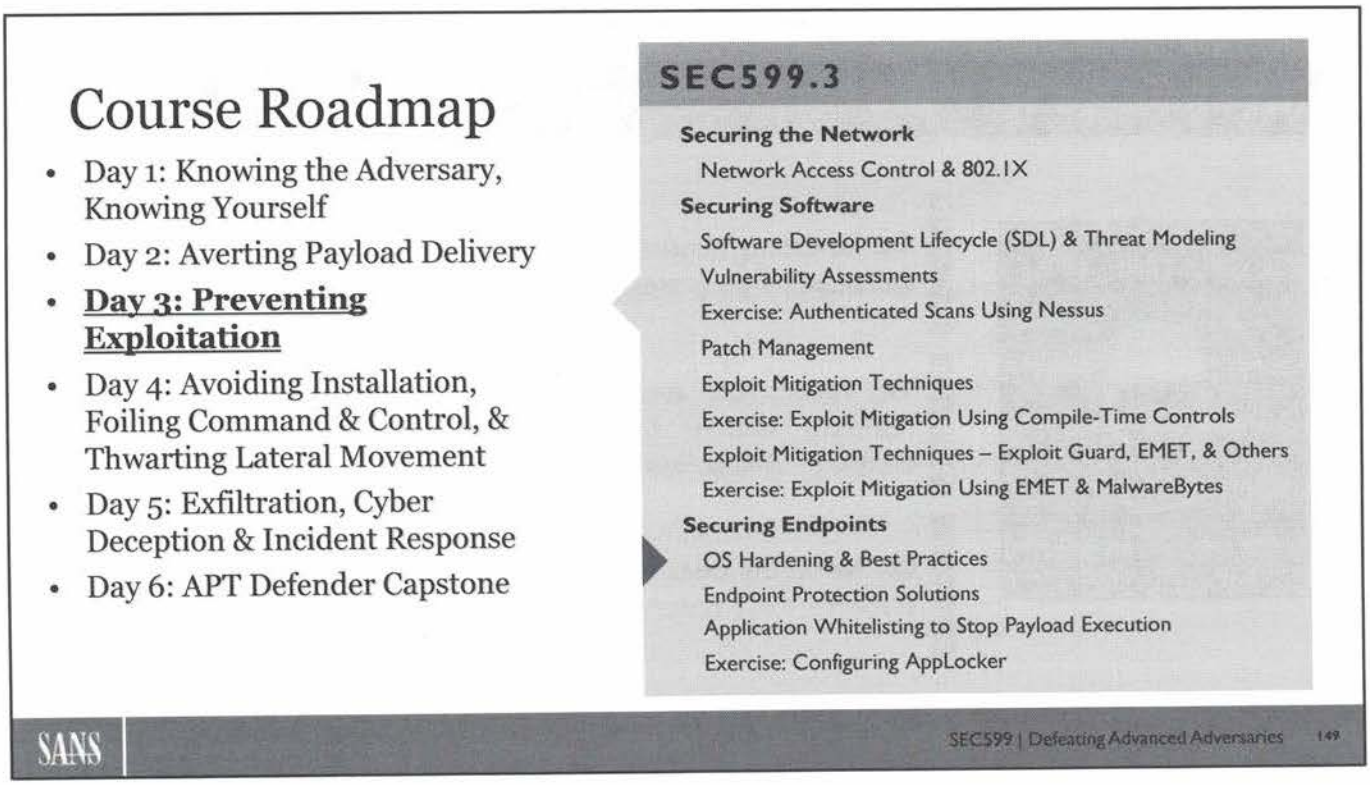


**Exercise – Hands-on Exploit Mitigation – Conclusion**

During the exercise, we used different ways of mitigating exploitation, even if a given application is vulnerable. Depending on your environment, any of these techniques could be beneficial, while a lot of them could already be implemented in your base OS.

It's important to note, that, as with many security issues, this is a cat-and-mouse game, and adversaries are increasing their efforts to overcome these exploit mitigation techniques, so we should not only rely on them to prevent exploitation!

For us defenders, it's a powerful plus to have an in-depth understanding of how current exploit mitigation techniques work, so we can make the right decisions on what type of protection measures we are to implement in what environment.




This page intentionally left blank.

## OS Hardening & Best Practices

OS hardening consists of configuring the operating system and installed programs to reduce the attack surface.

We reduce the attack surface by disabling options and removing features (change default passwords, uninstall / disable unused software…)

OS hardening does not include the installation of security software, like a host-based firewall.



**OS Hardening & Best Practices**

When we talk about OS hardening, we mean configuration of the operating system and installed applications. The practice of OS hardening does not involve the installation of additional security software, like a host-based firewall.

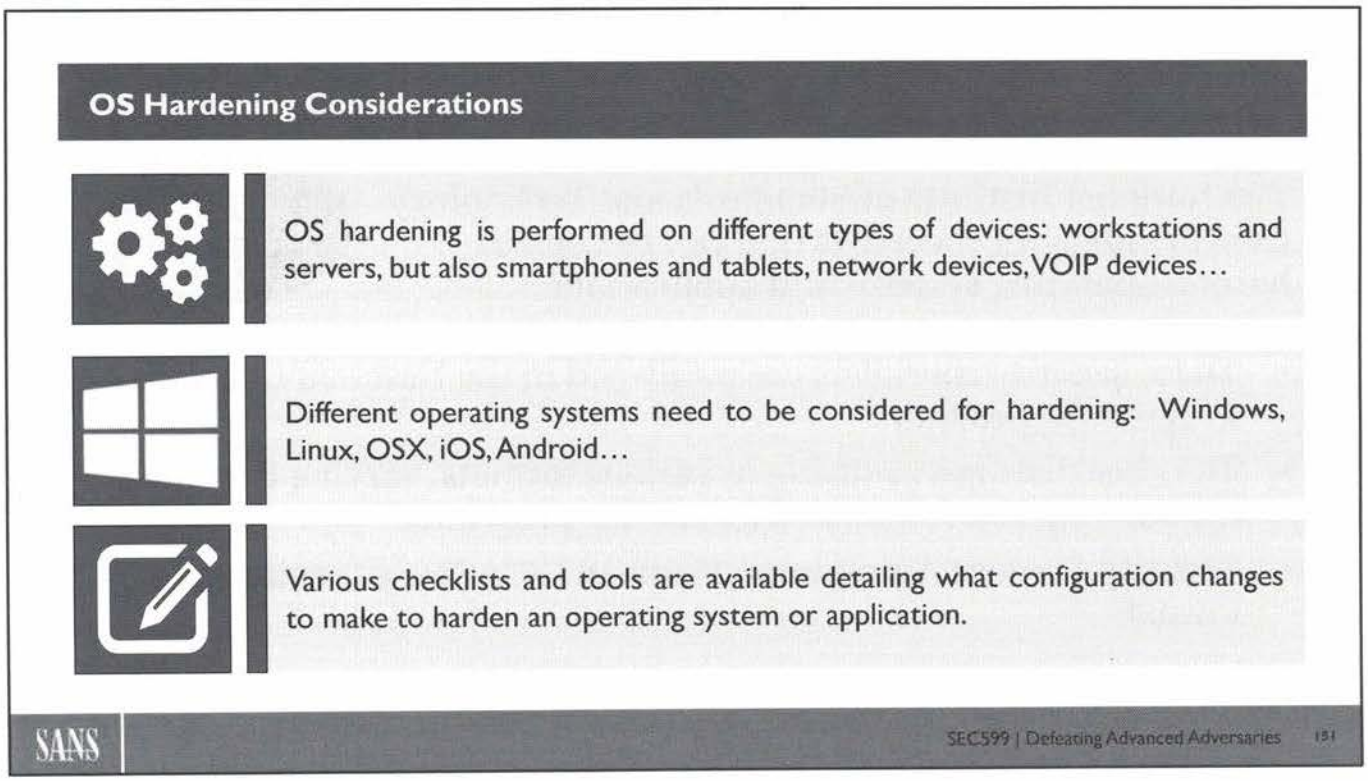
The operating systems we use in our enterprises (like Windows, Linux…) are general purpose operating systems. Such operating systems have many features to satisfy the needs of a wide variety of users. This implies that in our enterprise, many features are activated but not used.

The problem with active features is that they can be attacked. These features create a larger attack surface of our systems, and disabling or removing features reduces the attack surface.

Very old versions of Windows server operating systems used to install all features by default. For example, installing a Windows server with the default options would automatically install a web server and FTP server (IIS). Even if this web server and ftp server were never used, they were active and responded to queries over the network, making the server more vulnerable to attacks.

Nowadays, installation of Windows has changed: most features have to be activated explicitly.

OS and application hardening not only covers options and features to reduce the attack surface but also covers configuration of services to remove unauthenticated access and default credentials.




**OS Hardening Considerations**
OS and application hardening is not limited to computers.

It should be clear that we have to harden desktops, laptops, and servers. But these are not the only devices in our enterprise network that constitute our exposed attack surface.

We also have network devices, like switches, routers, wireless access points. Enterprise network devices have various options that can be configured and must be considered for hardening.

Other network connected devices are VOIP phones and network printers/scanners. These are commodity appliances that are often forgotten when it comes to security. VOIP phones and networked printers are often configured open so that they can serve as many clients as possible. But this enlarges the attack surface and should be reduced by hardening. As an example of a large attack surface, we want to mention that network printers often have an ftp server to facilitate printing of uploaded documents, and that such ftp servers have been used as pivot points in attacks.

Smart devices like smartphones and tablets should also be included in an OS hardening program.

OS hardening is not limited to Windows. Other operating systems also have features and options that dictate the attack surface, and that can be reduced by disabling features and options. Linux and OSX should be hardened for workstations, but operating systems for smart devices like iOS and Android should not be left out of the picture.

## NIST Checklist

The National Institute of Standards and Technology (NIST) publishes checklists that can be used to harden operating systems and applications:


- These security checklists are published in the National Checklist Program Repository
- The checklists are available in various formats, varying from text for humans to a formalized format for programs
- Security Content Automation Protocol (SCAP) is a formalized format



**NIST Checklist**

One of the most known providers of security checklists to harden operating systems and applications is the US National Institute of Standards and Technology: NIST.

This organization has a long tradition of publishing recommendations to configure operating system and applications, to make them harder to attack. This started with detailed instructions written in a guide that system administrators would read and apply to their system. To get an idea of the level of detail, such guides would mention registry keys and values to be configured in Windows.

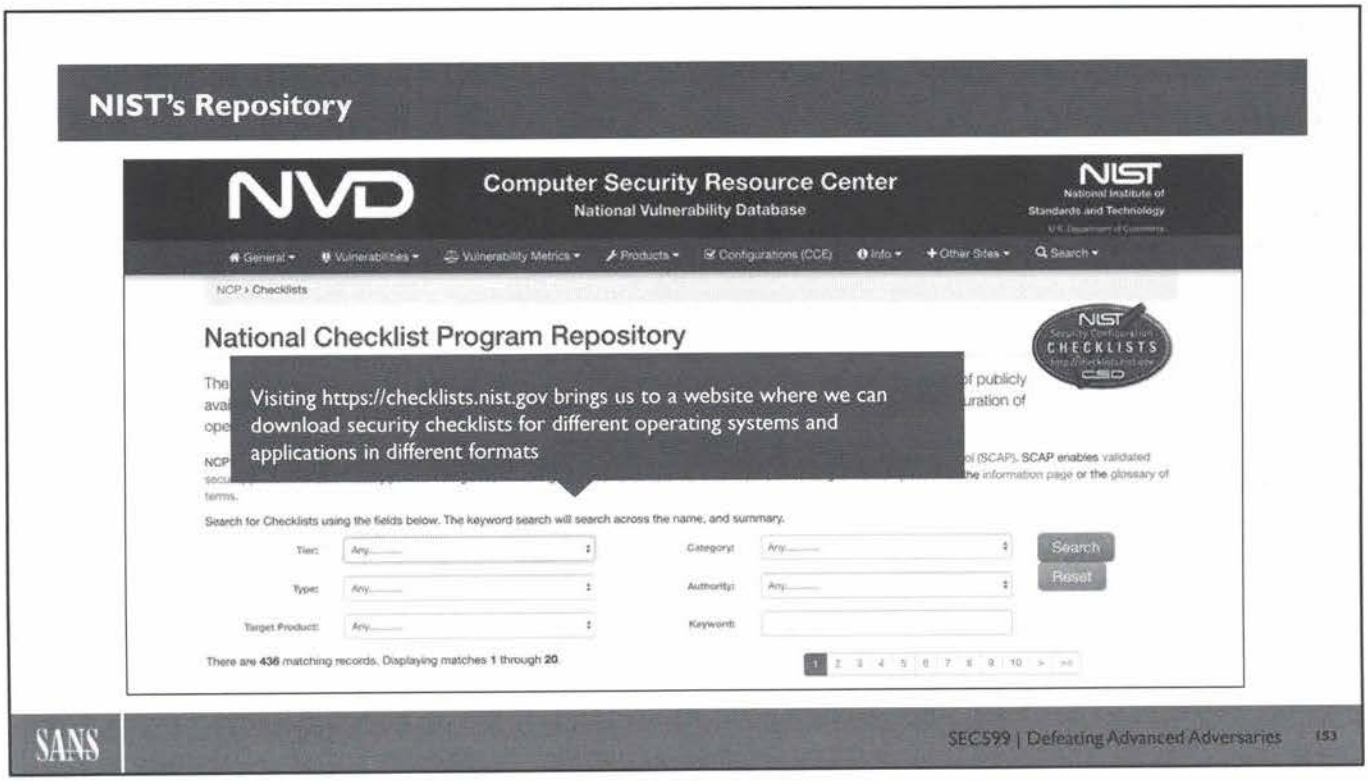
The security checklists are stored in the repository of the National Checklist Program, which can be accessed by going to https://checklists.nist.gov.

Providing guides written for humans was a good starting point, but due to the amount of technical detail in these guides, applying the recommendations to an operating system involved a lot of work that was error-prone. Scaling was another problem: applying the recommendations of a NIST checklist to a corporate network involved thousands of machines.

That is why NIST evolved its checklist repository to include formats that support automation: checklists can be downloaded in a machine-readable form, that can be applied by configuration applications to harden operating systems and applications.

The Security Content Automation Protocol (SCAP) is a format designed to automate vulnerability assessment and management. The NIST security checklists are available in SCAP format so that they can be used for automatic processing.

Checklists can be used to check the configuration of a system, and to change the configuration according to the recommendations of the checklist.




**NIST's Repository**
The security checklist repository of NIST can be accessed by visiting https://checklists.nist.gov.

As can be seen above, this will direct us to a website that enables us to specify the checklists we look for.

We select Target Product "Windows 10" as an example.
By clicking on the search button, we get 4 results.

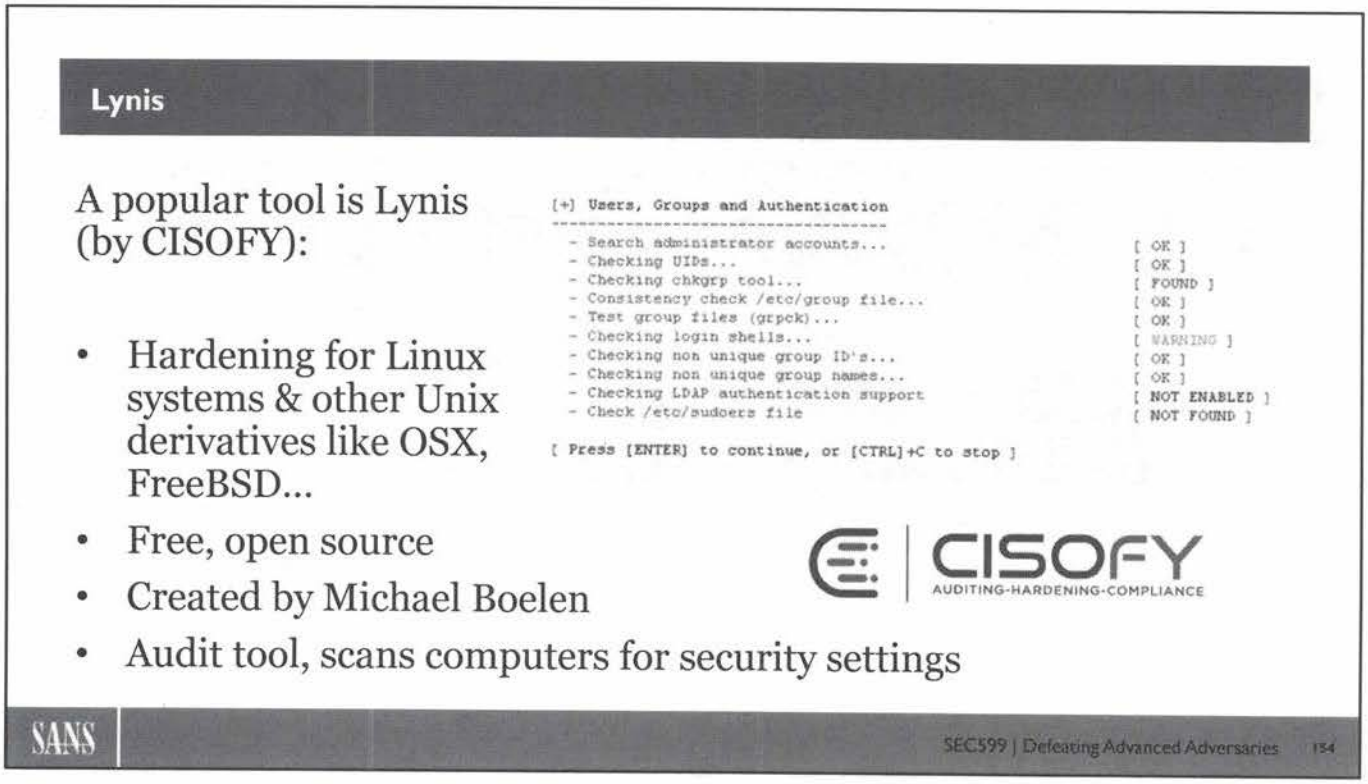
The first result is "Windows 10 STIG" produced by the Defense Information Systems Agency.

STIG stands for Security Technical Implementation Guide. It covers Windows 10 systems member of a domain and, therefore, covers Windows 10 Enterprise.

It supports the SCAP format, and also OVAL. This is an XML format meant for machine consumption, but it can also be rendered in human-readable form, although with a bit of practice, information can be obtained from its raw form. For example, this:

```
<criteria operator="AND">
    <criterion test_ref="oval:mil.disa.fso.windows:tst:388900" comment="Verifies Telnet Client feature is not installed" />
  </criteria>
```

With this XML chunk, we know that one of the checks is for the presence of Telnet. Telnet is a cleartext, unauthenticated remote access service, and should never be used.




**Lynis**

Besides NIST's security checklists, other methods are available to harden operating systems.

A popular tool for hardening Linux systems is Lynis. It is an open source tool (GPLv3 license) developed by Michael Boelen.

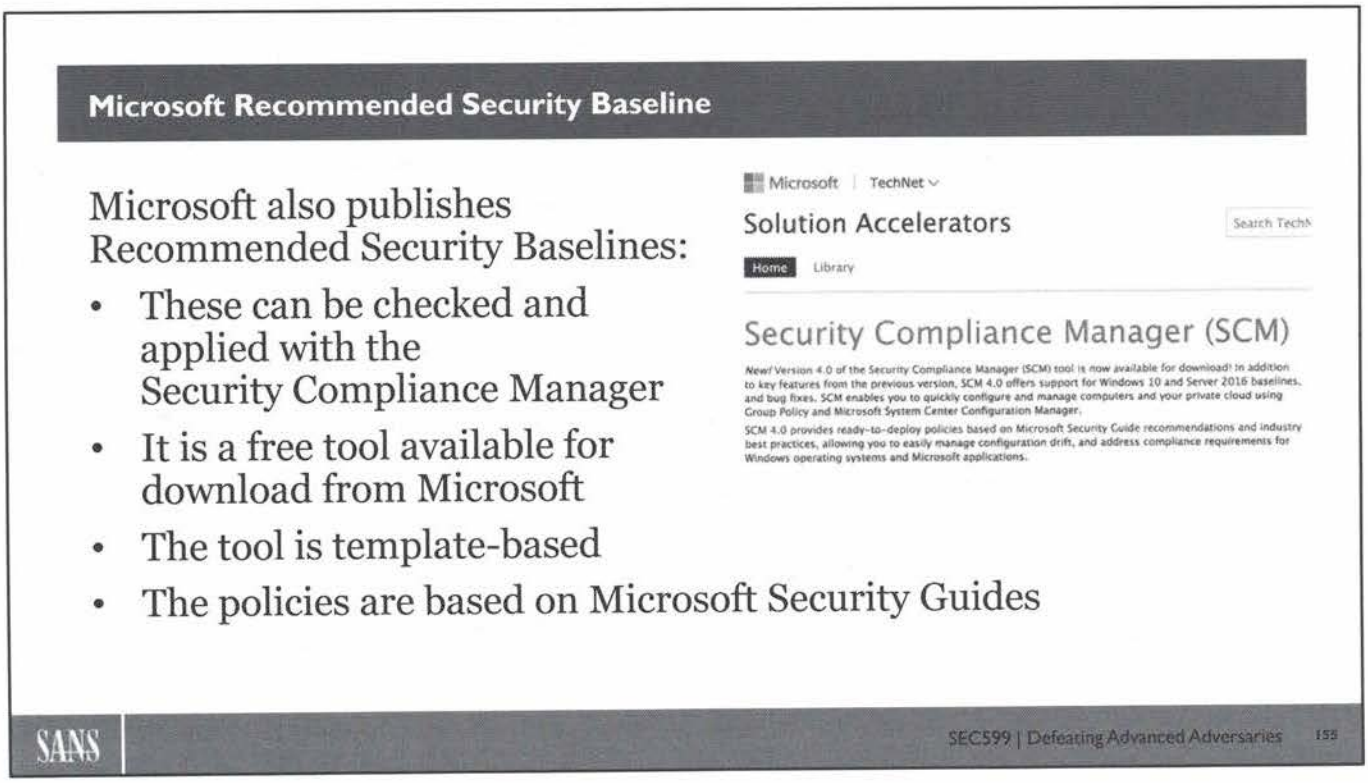
Lynis is an auditing tool, that will scan Linux computers for various security settings.
These include the following (non-exhaustive list):

- Kernel settings
- Authentication settings
- Installed daemons
- Installed applications
- Logging
- ...

It will also check the system for security issues and misconfigurations that can lead to compromise.

Lynis is also available for Unix-like operating systems like OSX, FreeBSD...

CISOFY also sells an enterprise version of Lynis with more scanning features and capabilities (like support for more than 10 clients).




**Microsoft Recommended Security Baseline**
Another source for guidelines to harden Microsoft Windows is Microsoft itself.

Microsoft publishes recommended security baselines and security guides.

These baselines can be automatically checked and applied with tools offered by Microsoft.
One of these tools is the Security Compliance Manager.

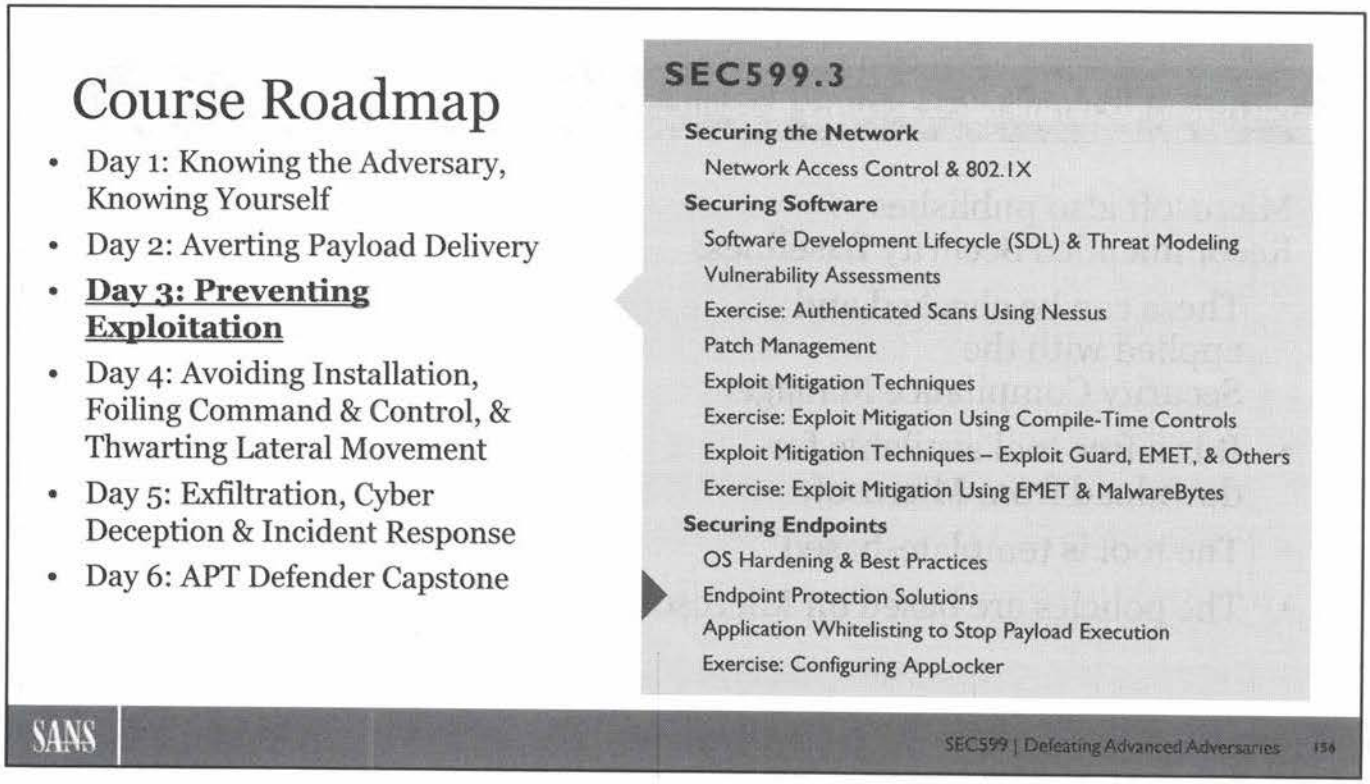
The SCM is a free tool from Microsoft. It is not part of a default Windows install but needs to be downloaded and installed.

SCM 4.0 supports the latest Windows versions (Server 2016 and Windows 10).

The tool not only allows managing Microsoft produced security baselines, but these can be modified to better suit your enterprise's environment. New templates can be created too, for custom configurations.

SCM can be used to deploy configurations to stand-alone machine too: machines that are not members of a domain.

Baselines can be exported to Microsoft supported formats like GPO policies, but also open formats like SCAP used by NIST, so that Microsoft Security Baselines can also be converted SCAP and audited with SCAP tools.




This page intentionally left blank.

## Endpoint Protection Solutions (1)

Endpoints are computer devices connected to the corporate network, like:

- Desktop computers
- Servers
- Laptops
- Smartphones
- ...

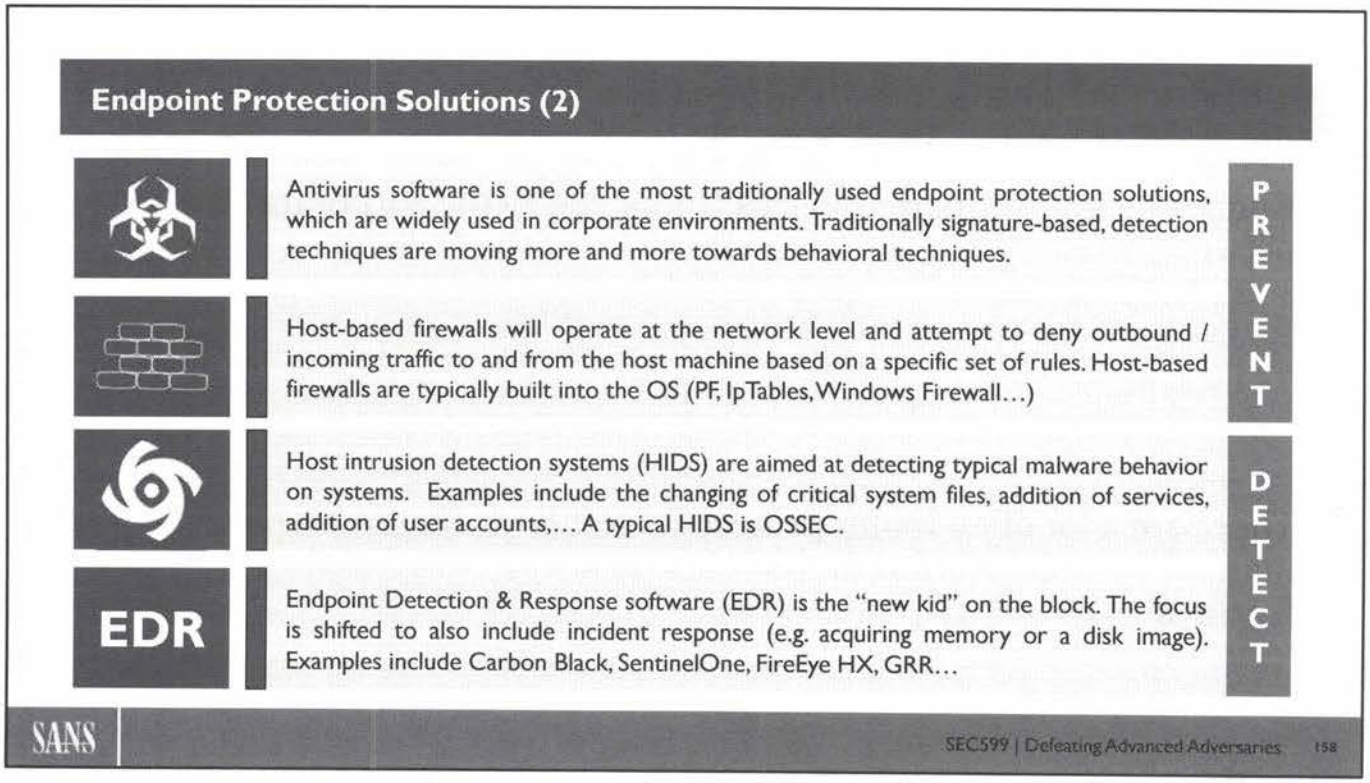
There are a lot of endpoint protection solutions on the market, many with widely different, but often also with overlapping, features.



**Endpoint Protection Solutions (1)**
An endpoint is a computer device that is a "network endpoint" on the corporate network: computer devices like desktop computers, laptops, smartphones, tablets…

As these devices are interesting to attackers to try to enter the corporate network, endpoint protection solutions have been developed. An endpoint protection solution is software running on the device to detect and prevent intrusions. Many vendors offer endpoint protection solutions, and the features they provide can vary widely.

We will use the term "features" because they can be stand-alone products or combined into one product.

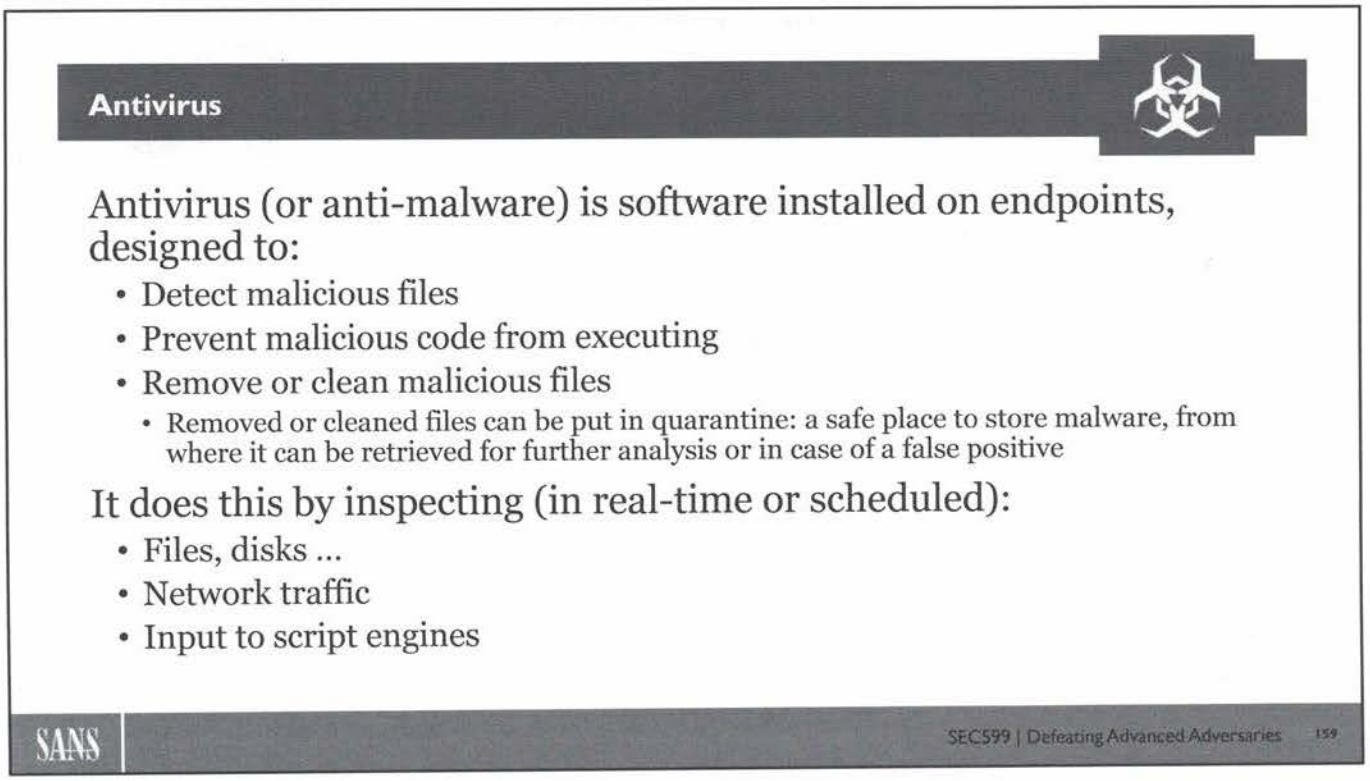



**Endpoint Protection Solutions (2)**

Different types of endpoint protection solutions exist, which often use an agent that is deployed on the endpoints. We use the following categories:

- Antivirus software is one of the most traditionally used endpoint protection solutions, which are widely used in corporate environments. Traditionally signature-based, detection techniques are moving more and more towards heuristic techniques.
- Host-based firewalls will operate at the network level and attempt to deny outbound / incoming traffic to and from the host machine based on a specific set of rules. Host-based firewalls are typically built into the OS (PF, IpTables, Windows Firewall…)
- Host intrusion detection systems (HIDS) are aimed at detecting typical malware behavior on systems. Examples include the changing of critical system files, addition of services, addition of user accounts… A typical HIDS (so focused on detection) is OSSEC, although using OSSEC's "active response" feature, it can also be used to actively react to detected issues.
- Endpoint Detection & Response software (EDR) is the "new kid" on the block. It's an "evolution" of HIPS also allowing active "response" to security issues (e.g. acquisition of a disk drive). Examples include Carbon Black, SentinelOne, FireEye HX, GRR…

It's important to note that this "abundance" of agents is leading to something the author likes to call "agent fatigue": organizations (& employees!) are reluctant to install 4 different agents for "security purposes". It's thus important to look for an "all-in-one" solution that can handle the different functions described above in one agent.




**Antivirus**

Antivirus, sometimes referred to as anti-malware, is software that is running on endpoints. Its goal is to detect malware and to prevent malicious code to execute.

Malware can be executable files (PE files), scripts, documents with scripts or documents with exploits and shellcode. Antivirus is mainly designed to detect malware in files.

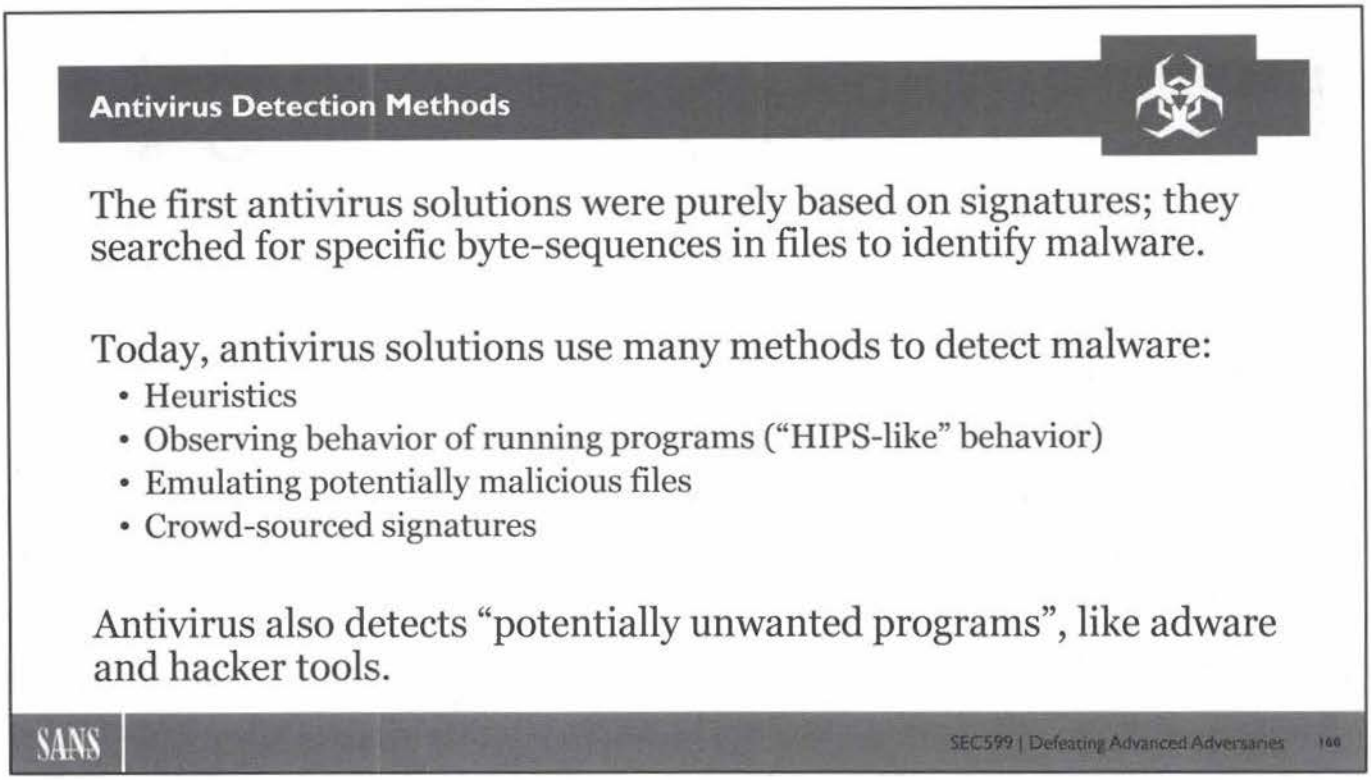
When antivirus detects malware in a file, it will block access to the file so that it cannot be opened for execution: the malicious code is prevented from executing. Besides blocking access to the file, antivirus will often remove or clean the file. Cleaning the files involves removal of malicious code: For example, with a virus that infects existing executables by appending the viral code to the end of the file, antivirus will clean the executable by removing the appended viral code and restoring the file to its original version.

As false positives can occur (antivirus wrongly identifies a benign file as malware), removed files and cleaned files are often quarantined: they are put in a safe place (a folder with encoded files) so that they can be restored later, should the detection turn out to be a false positive. Quarantined files are also useful for malware analysis.

Antivirus can do detection in real-time (on access scanning) or scheduled (on-demand scanning). On-access scanning means that files are scanned every time they are read or written. It is a good policy to have both on access scanning enabled and on-demand scanning scheduled (for example once a week outside working hours): this way, malware that was missed by on access scanning (for example because there were no signatures available to detect it) can get caught by on-demand scanning (with updated signatures).

Antivirus will scan files and disks, but a lot of antivirus products can also scan network traffic. There are even products that can scan the decoded content of TLS traffic by terminating this traffic with their own certificates.

On Windows, antivirus can also insert component to scan the input to script engines like VBScript and JScript.




**Antivirus Detection Methods**

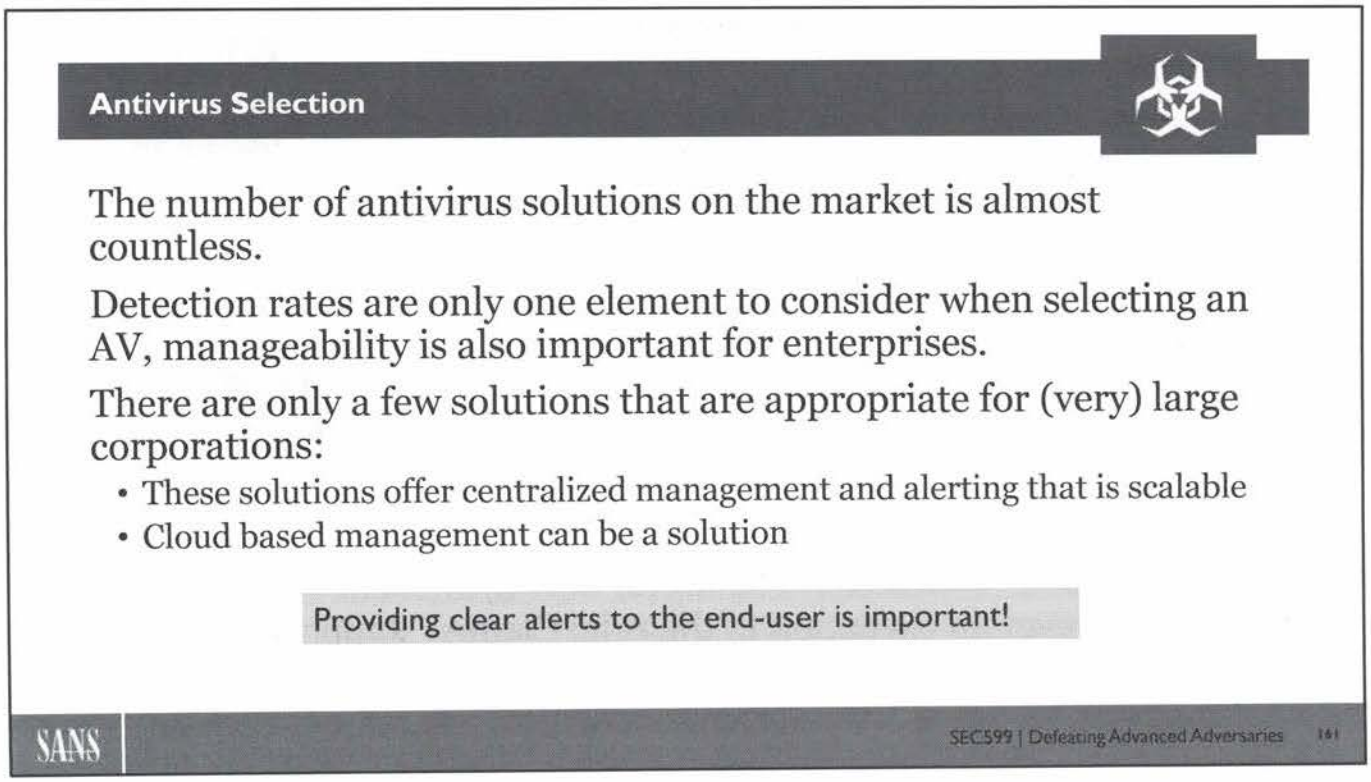
The very first antivirus programs were purely based on signatures: the searched through the content of files for specific byte sequences and identified files with these sequences as viruses. For example, a file is detected as the Stoned Virus by ClamAV antivirus if it contains the following bytes: 07 00 BA 80 00 CD 13 EB 49 90 B9 03 00 BA 00 01

Such signatures (and more complex types of signatures, with wildcards and regular expressions) need to be developed by the antivirus company and distributed to their clients. Signature database updates occur several times per day, due to the massive amount of malware in the wild. Signature-based detection offers good performance (it does not consume many computer resources), but it is reactive: only existing malware (or simple variants) can be detected with signatures.

That is why many alternative detection methods are implemented in modern antivirus programs:

- Heuristics: heuristics are simple rules-of-thumb. For example, an executables (PE file) is considered malware if it imports the Windows API calls to open processes, write to their memory and start executing code.
- Behavior observation: block a running program when it tries to execute a prohibited action. This is what we typically call "HIPS-like" behavior, as the AV engine will now act as a sort of Host-Based Intrusion Prevention engine.
- Emulation: before an executable is executed by the Windows operating system, the antivirus will emulate the execution of the file with its internal emulation engine, and observe its behavior. Based on this (emulated) behavior, the file is classified as malware or benign. As emulation takes many resources and time, a complete execution is not emulated, only the beginning
- Antivirus programs also exist in cloud-based versions: they have their signature database "in the cloud", and new signatures are automatically generated from client detections and shared via the cloud to all clients

Remark that antivirus not only classifies files as malicious or benign, but many also have a category called "potentially unwanted programs". These programs are not really malicious, but neither completely benign. "Hacker" tools fall into this category, like Metasploit and Nmap. Antivirus can help us detect advanced adversaries when they deploy such tools during lateral movement.




**Antivirus Selection**

The number of antivirus programs is staggering. VirusTotal "only" hosts 60+ different antivirus programs. This makes it hard to select the right antivirus solution for your enterprise.

There are specialized organizations and publications that will evaluate antivirus solutions on their detection rates. Simply put: how many files to they detect while scanning a set of malicious files? Many home users and companies make their choice based on these studies: they go for the antivirus with the best detection rates. But for enterprises, that produces often a sub-optimal solution. First of all, higher detection rates imply higher false positive rates. A false positive can be disruptive to the production environment of an enterprise (for example when a critical component of Windows is flagged as malicious, as has happened in the past with certain antivirus programs). Secondly, other criteria are important for an enterprise: Like, manageability and ease of integration.

For very large enterprises, there are actually not many appropriate antivirus solutions. When it comes to protecting hundreds of thousands of endpoints, many antivirus solutions cannot scale: centralized management fails, alerting is not reliable…

From a user awareness point of view, it is important to select an antivirus solution that provides clear alerts to the end-user (if possible tailored alerts). Centralized alerting is important for the SOC but end users must also be made aware of a malware detection on their endpoint so that they can change their behavior and take appropriate action.

## Antivirus Configuration

Antivirus requires configuration. It can be configured to:

- Configure real-time protection mode;
- Schedule scans;
- What type of files to scan;
- Configure exceptions of what not to scan;
- ...


Antivirus typically cannot be further adapted to include custom-made signatures, which could be useful during incident response. There is however an exception to the rule: ClamAV supports YARA rules!



**Antivirus Configuration**
Antivirus is an endpoint protection solution that can be deployed with default configurations, but often, for optimal performance and protection, the configuration needs to be tuned.

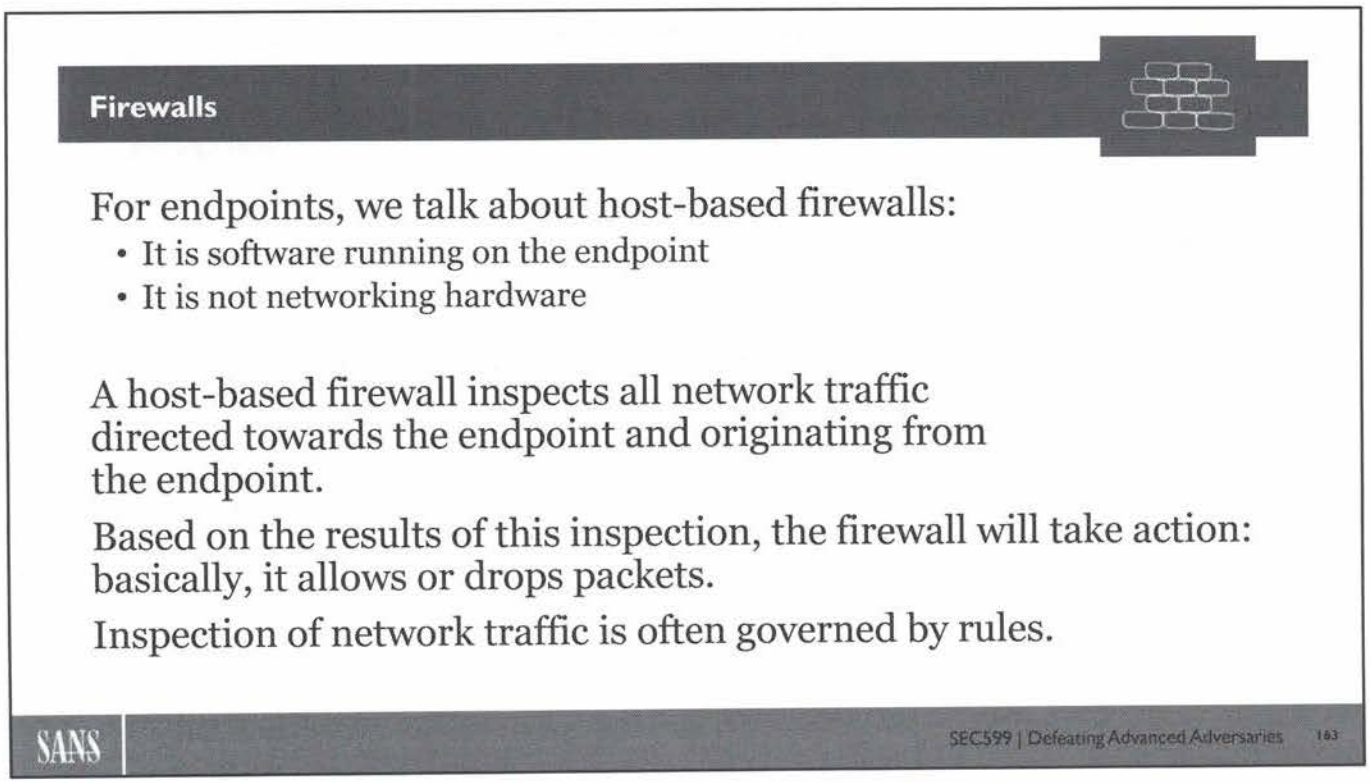
Depending on the performance impact on the endpoint, the antivirus might be configured to scan a set of files and not all files. For example, on a database server, we could exclude the database files from antivirus scanning.

On-access scan is highly recommended, but on-demand scan (scheduled scanning) should also be configured. Scheduled scanning means that at regular intervals, the complete disk of a machine will be scanned. This scheduled scan is best preceded by a signature database update so that the latest signatures are used. A scheduled scan can detect malware that was missed by on access scans.

Antivirus can also be configured for the actions it has to take when malware is detected:

- Produced a centralized alert for the Security Operation Center
- Warn the end-user
- Delete the file
- Clean the file
- Quarantine the file
- ...

During incidents, it would be very useful for incident responders to be able to run scheduled scans with custom-made signatures. Unfortunately, this type of configuration is often not possible with antivirus products. They signature database have a proprietary and protected format and are not open. Sometimes, (paid) assistance from the antivirus vendor is available to create and deploy custom signatures.




**Firewalls**

When we talk about firewalls on endpoints, we mean host-based firewalls. Not hardware in or before the endpoint, but software deployed on the endpoint operating system.

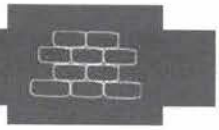
Like a hardware firewall, a host-based firewall inspects network traffic and can block network traffic (drop packets). It looks at all ingress and egress traffic of the endpoint, and contrary to a hardware firewall, it can also link network activity to specific programs running on the operating system.

For example, a host-based firewall will not only be able to inspect SMTP traffic (TCP connections to port 25) but also link it to the Outlook process.

Network traffic will be allowed or blocked when coming into the endpoint or when leaving the endpoint, based on the result of the network traffic inspection. Inspecting network traffic is often done with rules. Firewalls come with a set of predefined rules and can be extended with new rules. For example, a firewall can come with rules that block all SMTP traffic, except when it originates from the Outlook program. This rule is an example of a rule that is often used to combat SPAM-sending malware.

A host-based firewall can also block all incoming traffic when the endpoint is connected to an untrusted network, like public Wi-Fi. Technically speaking, all incoming SYN packets will be dropped, preventing access to open ports on the operating system.

## Windows Firewall



Windows includes a built-in firewall.

Introduced with Windows Vista, the Windows Filtering Platform is used to provide firewall functionality.

The Windows firewall uses the Windows Filtering Platform.

Many vendors offer a firewall for Windows:

- When such a firewall is installed, the built-in Windows firewall is disabled
- Some of these solutions do not install their own firewall engine but use the Windows Filtering Platform

The Windows firewall comes with many predefined rules; the creation of custom rules is supported.



**Windows Firewall**

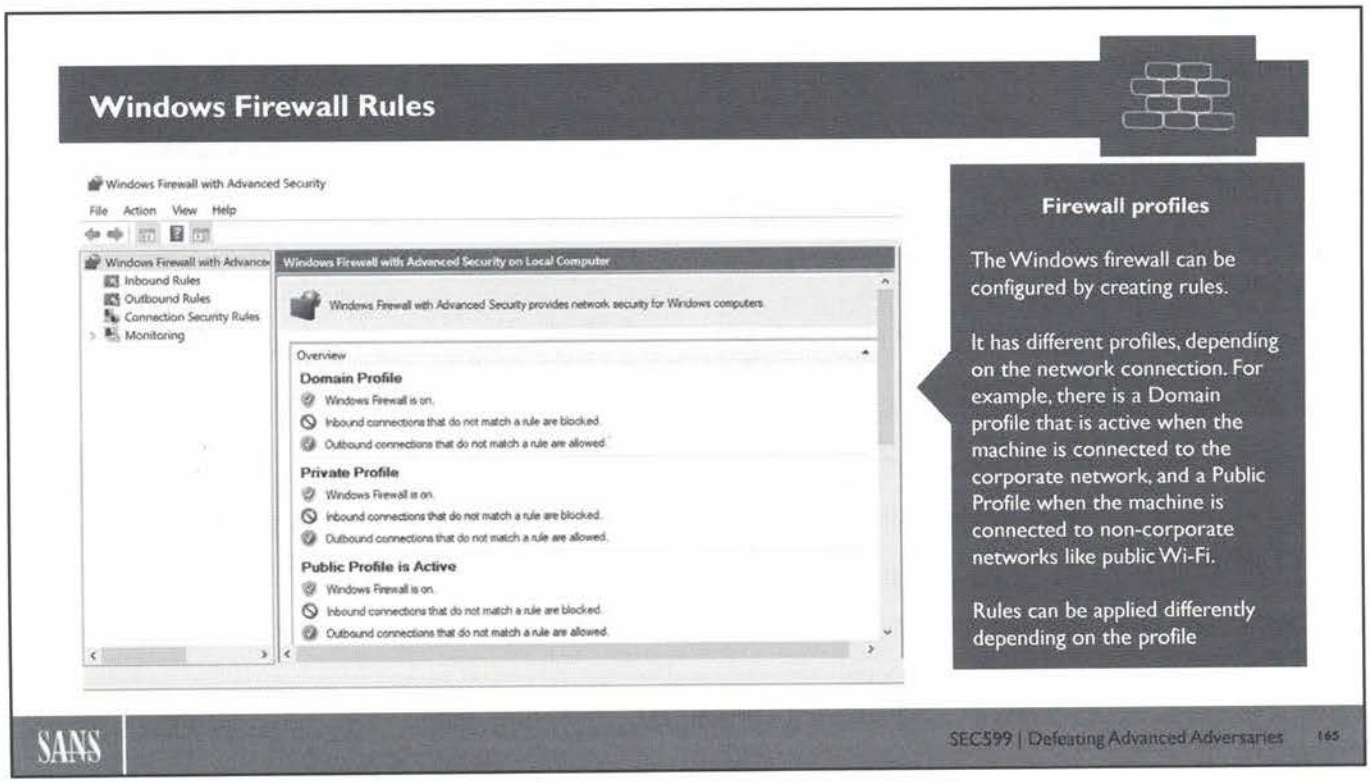
The Windows operating system comes with a built-in host-based firewall. Microsoft started to include a firewall with Windows XP SP2, because of the frequent, worldwide incidents with worms (malware that replicates automatically between hosts over a network, like the Internet). This was meant as mitigation, but was not an ideal technical solution, because it was not designed from the start.

With Windows Vista, Microsoft engineers redesigned the network stack and included features for firewalls in their design. The Windows firewall introduced with Windows Vista is a new product, and it uses the new Windows Filtering Platform. This allows for a host-based firewall that is properly integrated into the Windows operating system.

There are other solutions on the market than the built-in Windows firewall. Several vendors developed their own host-based firewall products: when this solution is installed on an endpoint, the built-in Windows firewall is disabled, and its functions are taken over and extended by the installed firewall solution.

Not all third-party solutions have developed their own firewall engine. Some of these solutions don't install their own firewall engine but use the Windows Filtering Platform. They replace the existing Windows firewall user interface with their own interface and supplement the platform with extended features.

The built-in Windows firewall is rule-based, and comes with many predefined rules. The creation of custom rules is also supported, as we will see in the next slides.




**Windows Firewall Rules**

Using the configuration tool "Windows Firewall with Advanced Security", it is possible to view and manage the existing rules and to create new rules.

This tool groups rules in inbound rules and outbound rules. Rules (pre-defined and custom) can be enabled or disabled on a rule-per-rule basis.

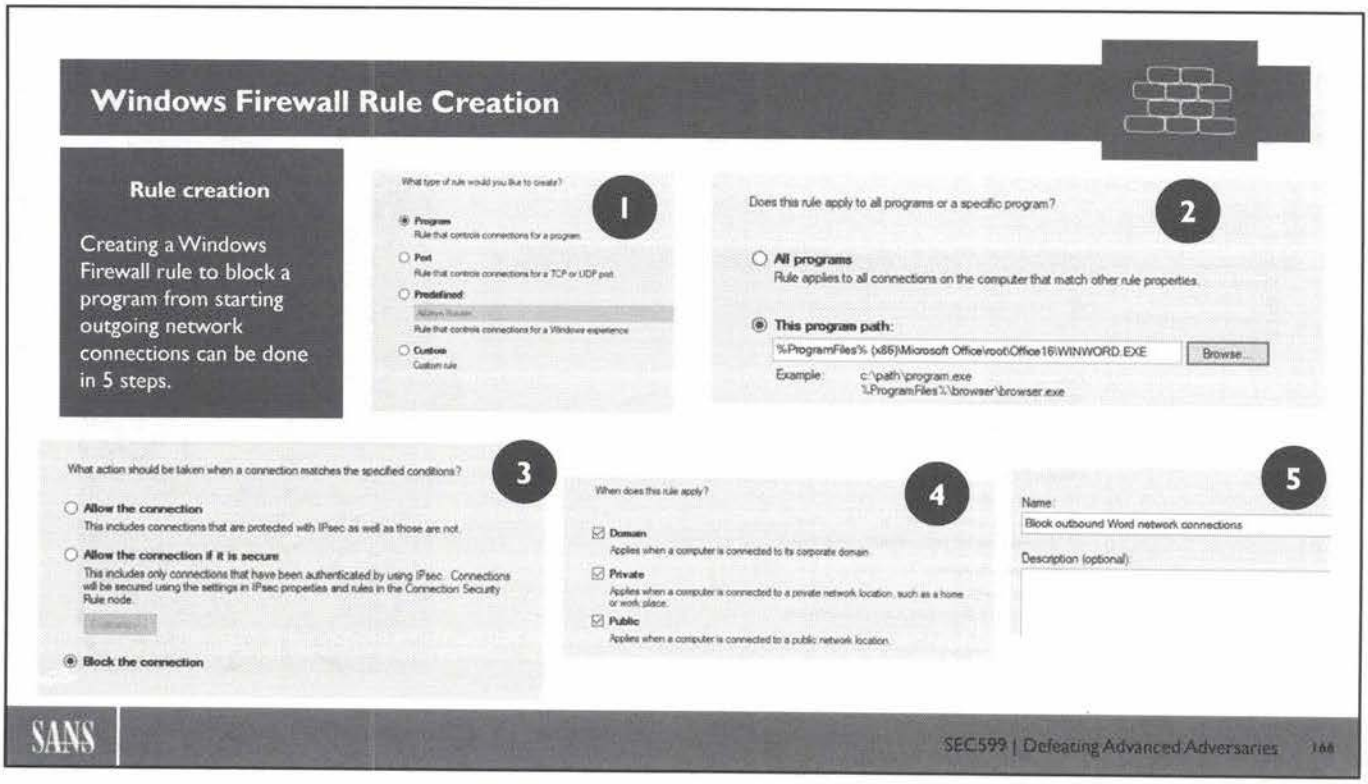
The Windows firewall uses different profiles, depending on the type of network connection:

- Domain profile: This profile is active when the Windows machine is connected to the corporate network with domain controllers
- Private profile: This profile is active when the Windows machine is connected to a non-corporate, private network: For example, the user's home network
- Public profile: This profile is active when the Windows machine is connected to a non-corporate, public network: For example, public Wi-Fi

Rules can also be activated per profile. The domain network is considered as trusted, and fewer rules are activated for this profile than for less trusted networks like private networks and hostile networks like public Wi-Fi.

By default, profiles have:

- Default deny for all inbound connections: inbound connections that do not match a rule are blocked
- Default allow for all outbound connections: outbound connections that do not match a rule are allowed




**Windows Firewall Rule Creation**

In this example, we create an outbound firewall rule to prevent Word from accessing the network. This rule will prevent malicious Word documents from downloading their payload from the Internet, as this requires the Word process to initiate TCP connections. It will not prevent Word from accessing network shares, as the necessary network connections to access network shares are performed by the Windows operating system and not the Word process.

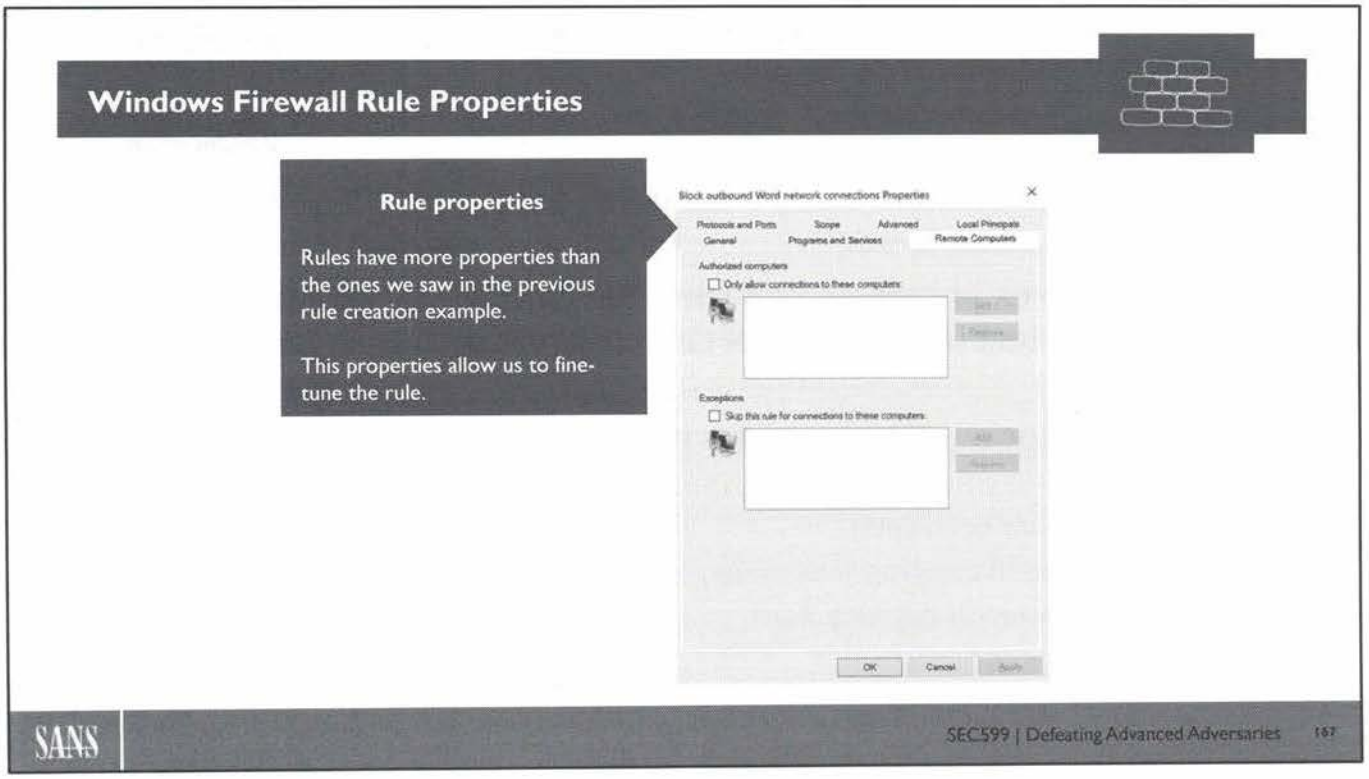
First, we need to decide which type of rule to create: we create a rule for a particular program.

In the second dialog, we select the Word program: winword.exe

In the third dialog, we can select the action: allow the connection or block the connection. In this example, we want to block the connection.

For the fourth step, we select the profiles for which this rule will be activated: we want to block Word from connecting to the network in all cases, so we select all profiles.

Finally, we name the rule and can add a comment.

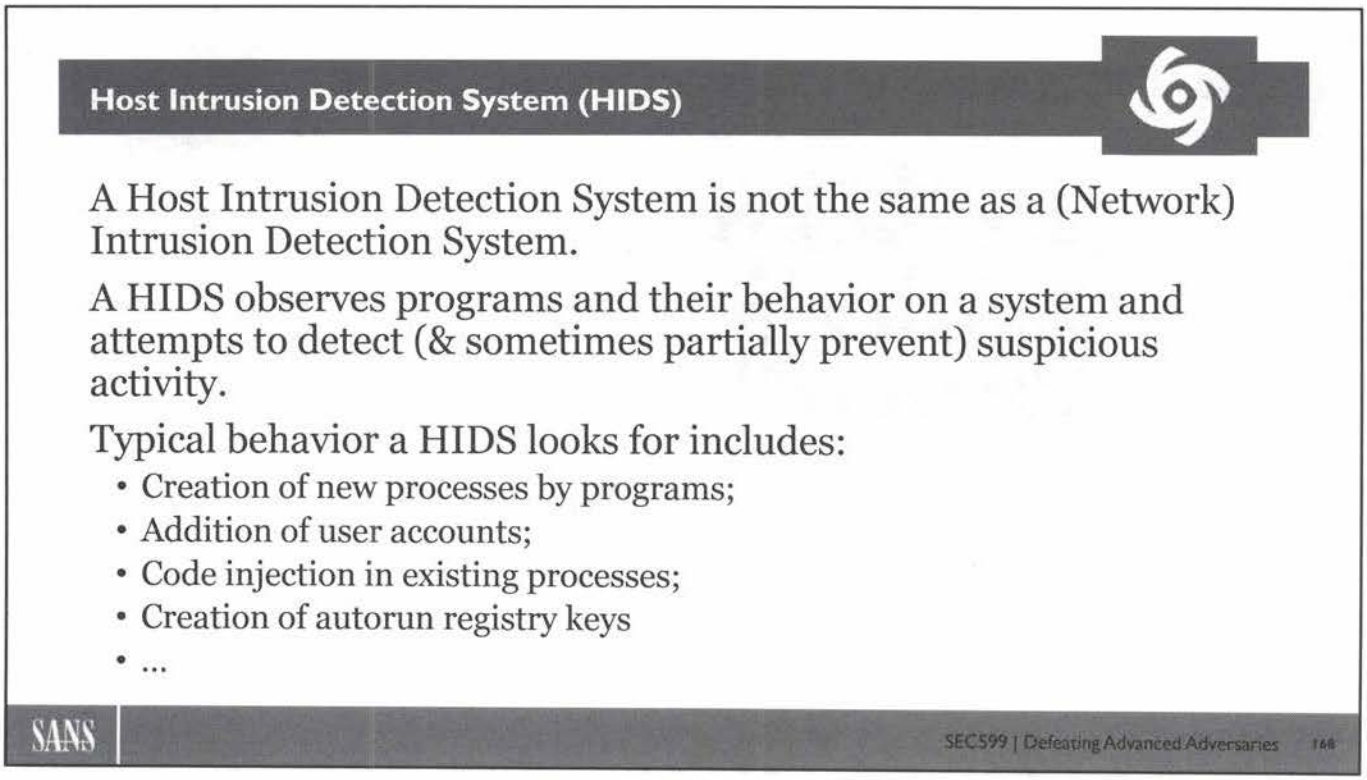



**Windows Firewall Rule Properties**

Rules have more properties than the ones we saw in the previous rule creation example. If we open the rule, we will see more tabs with properties than the dialogs we filled out with the wizard.

We can further specify conditions for a rule, allowing us to further refine the rule.

The example above shows the Remote Computers tab: with this tab, we can add exceptions to our rule: we can specify computers to which Word would be allowed to connect, despite our blocking rule. This comes in handy if we have valid business reasons to allow Word to connect to internal machines; for example, a database server from which information has to be retrieved to create monthly reports with Word macros.



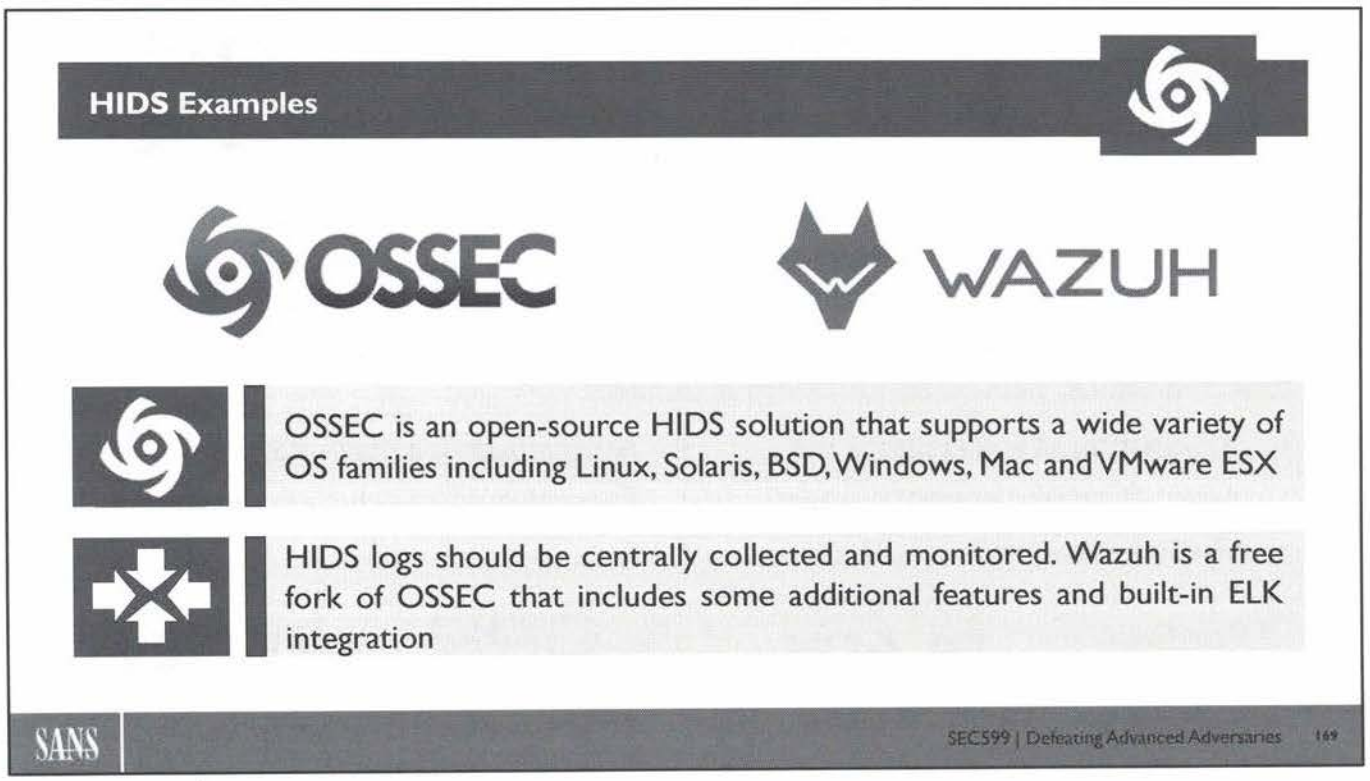

**Host Intrusion Detection System (HIDS)**
A host intrusion detection system is not network-based, it is, in fact, a very different thing. Unlike a classic (network) IDS, it typically runs as an agent deployed on the endpoints and inspects processes (running programs), file system activity, memory…

A HIDS inspects processes and their behavior when interacting with system resources and other processes, and it can detect / trigger on certain actions by these processes if they exhibit malicious intent. Some typical actions a HIDS will typically look out for include:

- A process starting a new program
- A process accessing the memory of another process
- Installation of drivers
- Termination of processes
- Injection of code
- Creation of autorun registry keys (to achieve persistence)
- …

This rule cannot be applied indiscriminately because many legitimate programs perform these actions as well (e.g. EMET & AV engines typically rely on DLL injection as well). To handle these exceptions, HIDS vendors have different solutions. There are HIDS with extensively pre-defined rules, that form a sort of whitelist of processes that are allowed these types of behaviors.

There are also self-learning HIDS systems: when first installed on a system, they are placed in learning mode and just observe the behavior of the processes on the system. Everything that happens in learning mode is considered as normal, and will not be alerted upon when learning mode is switched of subsequently.

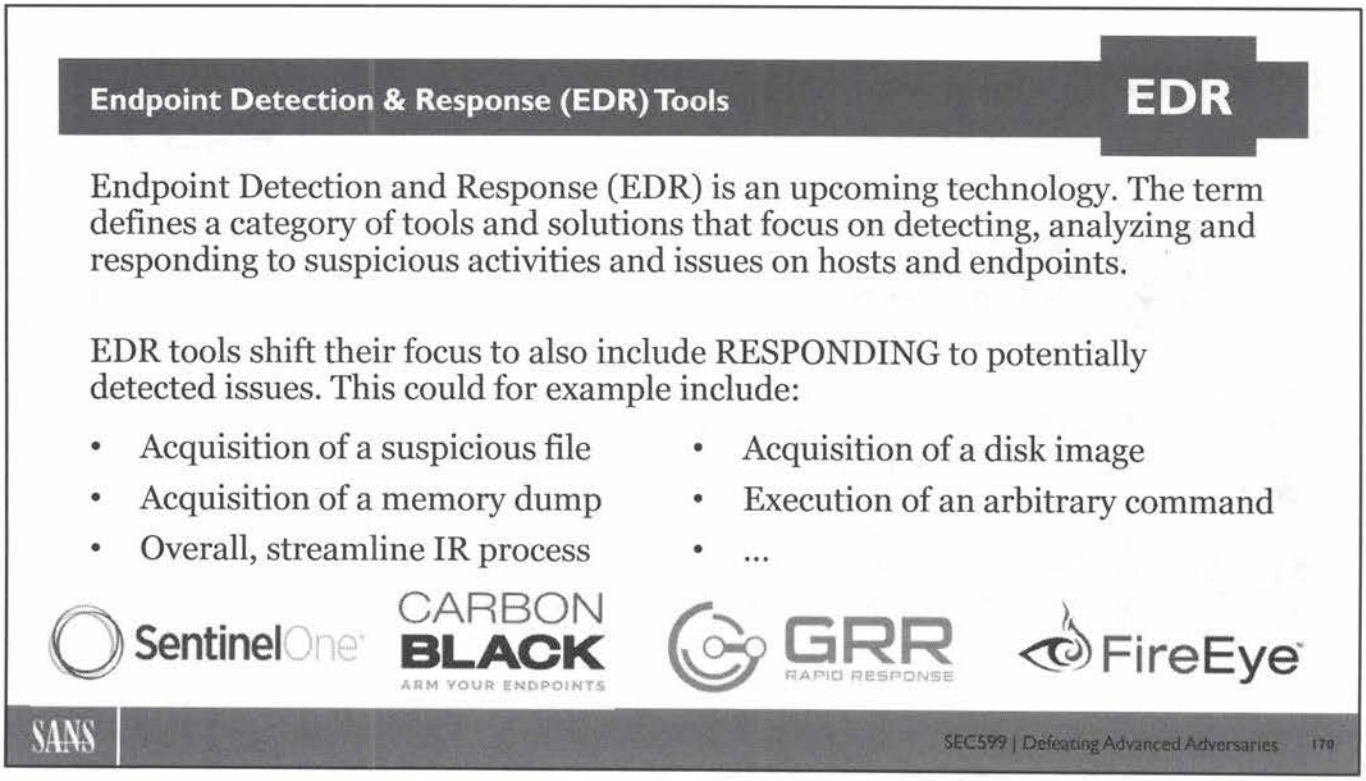



**HIDS Examples**

HIDS software can be very useful to supplement antivirus solutions: they can catch malware or unwanted software that antivirus engines would not block (or even detect) due to a lack of signatures. Advanced adversaries will often target their malware and customize it for your corporate environment so that signature-based systems would not block it. HIDS solutions have a better chance of detecting "unknown" malware, as they focus strongly on analyzing the behavior of the system and detect malicious behavior.

An interesting example of a HIDS (so focused on detection) solution is OSSEC, which is an open-source software that supports a wide variety of OS families including Linux, Solaris, BSD, Windows, Mac and VMware ESX.

As with any security tool, it's a good strategy to centrally collect and monitor HIDS logs. Wazuh is a free fork of OSSEC that includes some additional features and built-in ELK integration. Due to their "behavioral" nature, it's beneficial to have experienced personnel review the HIDS logs for anomalies and possible incidents.



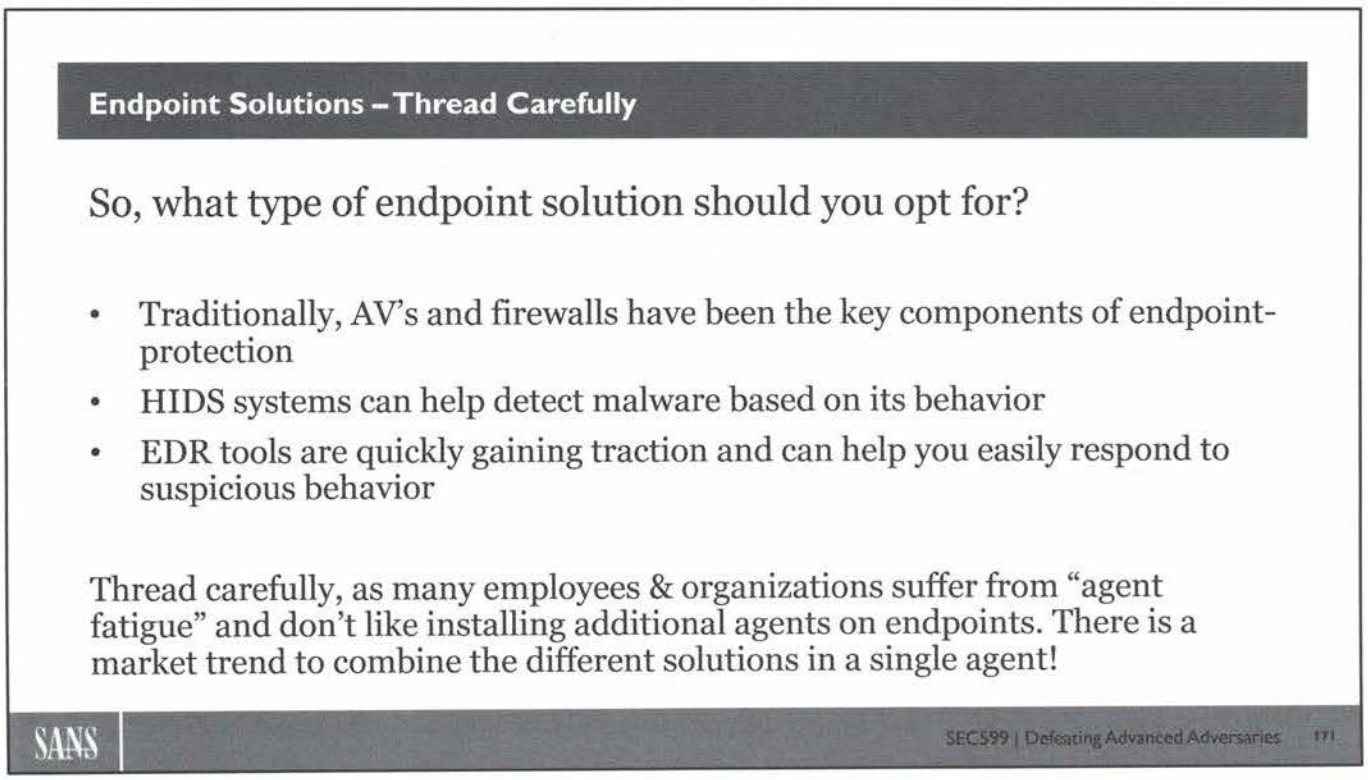

**Endpoint Detection & Response (EDR) Tools**

Endpoint Detection and Response (EDR) is an upcoming technology. The term defines a category of tools and solutions that focus on detecting, analyzing and responding to suspicious activities and issues on hosts and endpoints. EDR tools operate by monitoring endpoint and network events and centrally storing these logs & information to perform further analysis, reporting & response. As with most other endpoint solutions, a software agent installed on host systems is required to perform monitoring, response, and reporting.

The key difference between the previously mentioned endpoint solutions and EDR tools is that EDR tools shift their focus to also include RESPONDING to potentially detected issues. This could for example include

- Acquisition of a single file (e.g. a malware sample for further analysis)
- Acquisition of a memory dump / hard disk image for further forensic analysis
- Running arbitrary commands on the affected system;
- ...

Some examples of EDR tools include CarbonBlack, SentinelOne, GRR, FireEye HX... Furthermore, many AV vendors are adding EDR-like functionality as an optional component to their AV engines.




**Endpoint Solutions – Thread Carefully**
So, what type of endpoint solution should you opt for?

- Traditionally, Antivirus solutions and firewalls have been at the core of endpoint-protection solutions;
- HIDS systems can be a useful supplement to detect malware / suspicious software based on its behavior;
- EDR tools are quickly gaining traction and can help you easily respond to suspicious behavior;

The selection of the type of solution will very much depend on the maturity of your organization:

- If you have no SOC (or any other team) that will analyze the logs, a HIDS system won't truly realize its potential in your environment;
- Likewise, if you are not ready to start doing incident response, you won't be able to fully leverage an EDR tool.

Tread carefully, as many employees & organization suffer from "agent fatigue" and don't like installing additional agents on endpoints. There is a market trend to combine the different solutions in a single agent!

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

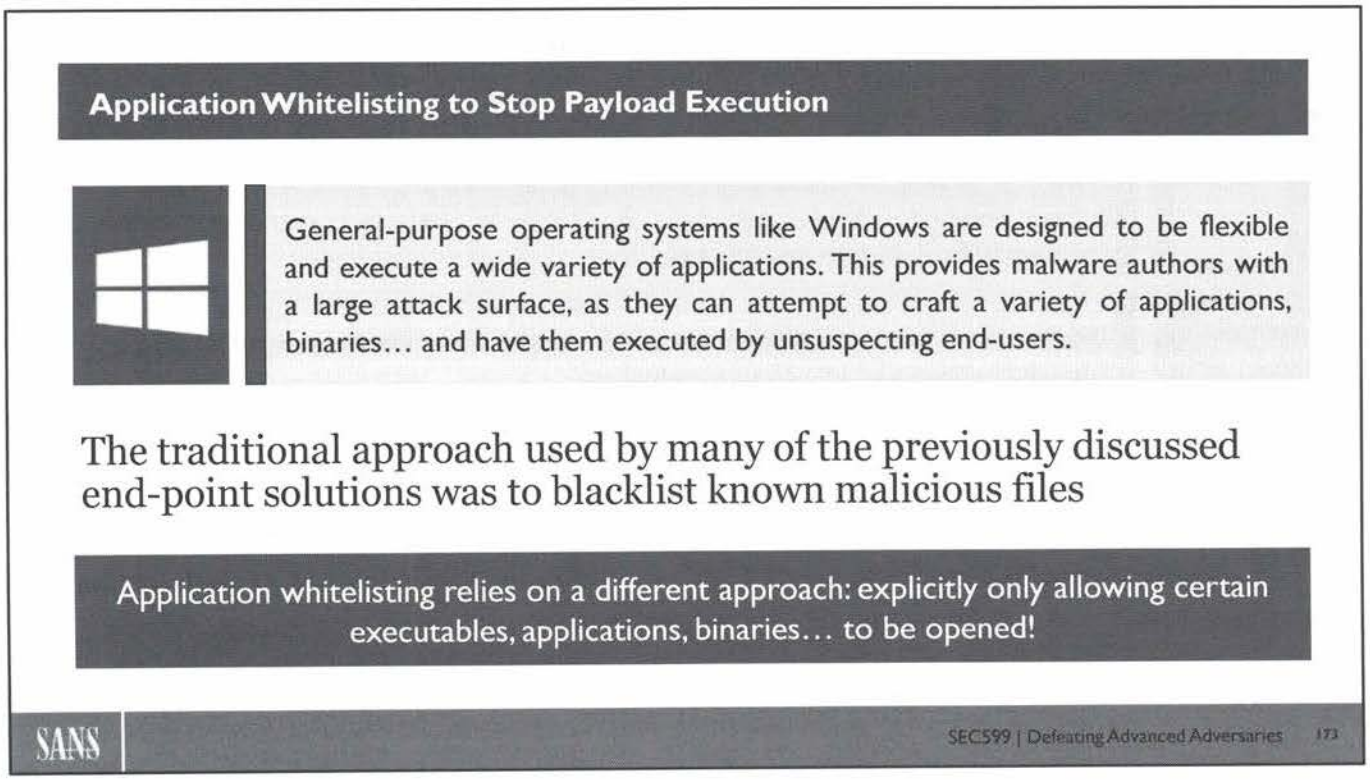
**Securing the Network**
- Network Access Control & 802.1X

**Securing Software**
- Software Development Lifecycle (SDL) & Threat Modeling
- Vulnerability Assessments
- Exercise: Authenticated Scans Using Nessus
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation Using Compile-Time Controls
- Exploit Mitigation Techniques – Exploit Guard, EMET, & Others
- Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**
- OS Hardening & Best Practices
- Endpoint Protection Solutions
- Application Whitelisting to Stop Payload Execution
- Exercise: Configuring AppLocker



This page intentionally left blank.

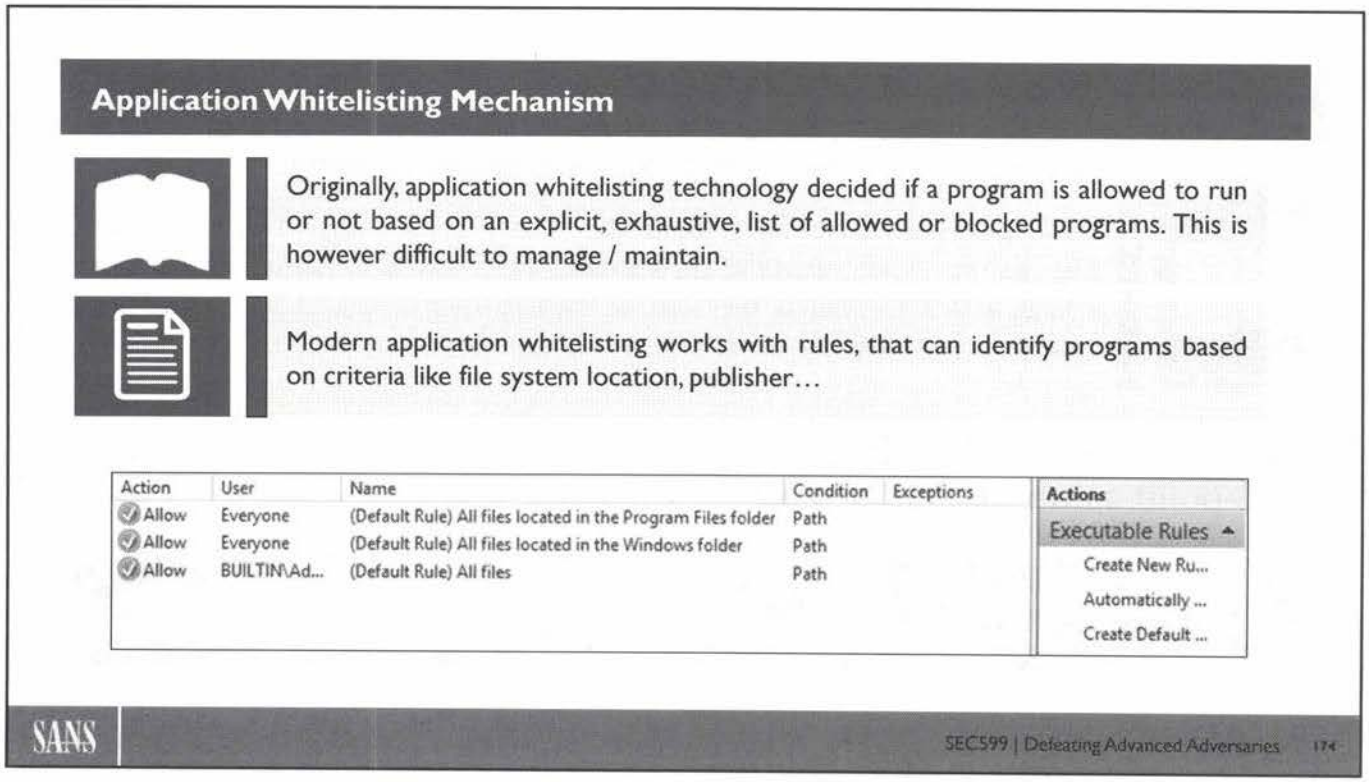



**Application Whitelisting to Stop Payload Execution**

General-purpose operating systems like Windows are designed to be flexible and execute a wide variety of applications. The flexibility of general-purpose computers and general purpose operating systems is an advantage for companies: the same type of computers and OS can be used to support many activities. This provides malware authors with a large attack surface, as they can attempt to craft a variety of applications, binaries… and have them executed by unsuspecting end-users.

Mobile devices are generally more restrictive: An example of an operating system that is considered not to be general purpose is Apple's iOS operating system, designed to run on iPhones and iPads. iOS does not support the execution of all types of programs: besides programs developed by Apple, only applications (apps) are allowed to run on iOS. Apps are restricted in their interaction with OS resources.

The traditional approach used by many of the previously discussed end-point solutions was to blacklist known malicious files. As we've seen, however, this has proven to be ineffective due to the rapid speed at which new malware samples are appearing. Application whitelisting relies on a different approach: Explicitly only allowing certain executables, applications, binaries… to be opened!




**Application Whitelisting Mechanism**

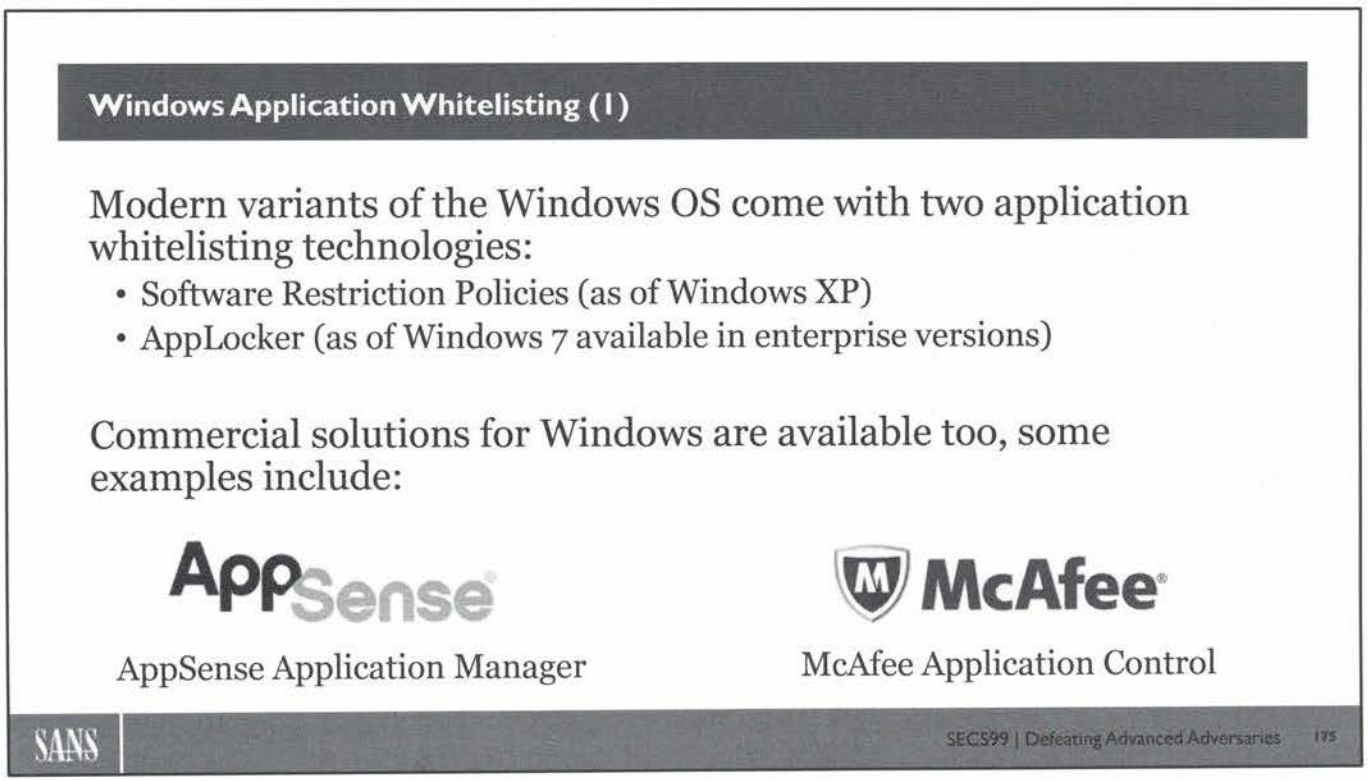
Originally, application whitelisting technology decided if a program is allowed to run or not based on an explicit, exhaustive, list of allowed or blocked programs. This is however difficult to manage / maintain.

Modern application whitelisting technology takes another approach: rules are used to decide if a program is allowed to run or not. A rule uses criteria to identify to which programs it is applied. Criteria can be:

- File name or file path
- Digital signature certificates
- Hashes
- ...

Rules can be used to whitelist programs or blacklist programs. In a whitelisting approach, the convention is to block all programs except the ones explicitly allowed by rules. A blacklisting approach is the opposite: All programs are allowed except programs explicitly identified by rules.

In general, a whitelisting approach is preferred over a blacklisting approach, as blacklisting is more brittle: A single program that is not blacklisted can be used to compromise the system.




**Windows Application Whitelisting (1)**
To implement application whitelisting on Windows, we have essentially 2 choices: We can use built-in application whitelisting technology, or we can use third-party technology.

Windows has two application whitelisting technologies: Software Restriction Policies and AppLocker. AppLocker was introduced after Software Restriction Policies. Software Restriction Policies are an older application whitelisting technology that is still available on Windows. If you have the choice, we recommend using the modern AppLocker approach.

AppLocker is completely implemented in the Windows kernel, while Software Restriction Policies has components in the kernel and components in userland. Userland components are more prone to tampering.

Many software security companies offer application whitelisting technology too. As an example, we mention 2 third-party application whitelisting technologies: AppSense Application Manager and McAfee Application Control.

AppSense creates software to manage Citrix servers. AppSense Application Manager allows controlling which applications are executed and which are blocked on Citrix servers.

McAfee Application Control is an addition to McAfee's endpoint security solution: it's application whitelisting technology.

In general, third-party solutions offer more features and management options than Microsoft's solutions.

## Windows Application Whitelisting (2)

- Software Restriction Policies were introduced with Windows XP, and are still available on the latest Windows versions.
- AppLocker was introduced with Windows 7, and is only available on "enterprise" versions of Windows.
- Software Restriction Policies and AppLocker are controlled through local or active directory group policies.

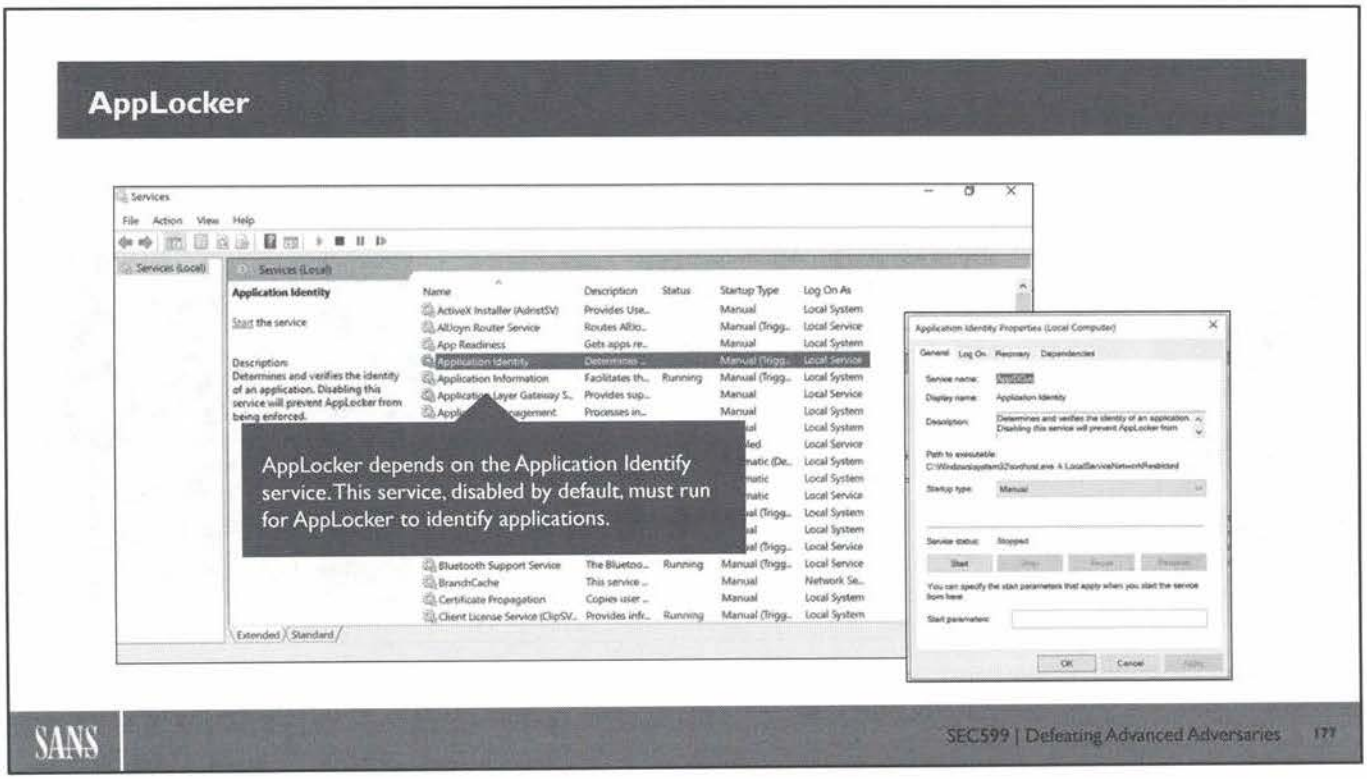


**Windows Application Whitelisting (2)**
Microsoft application whitelisting technologies come shipped with Windows, but depending on the Windows version, they can be unavailable.

Software Restriction Policies were introduced with Windows XP. Although it is the older technology, replaced by AppLocker, it is still available on the most recent Windows versions like Windows 10.

AppLocker was introduced with Windows 7. It can only be used on "enterprise" versions of Windows, and not on consumer versions of Windows. For example, on Windows 10, AppLocker is available with Windows 10 Enterprise, but not with Windows 10 Pro. We refer to Microsoft for a list of supported versions.

Software Restriction Policies and AppLocker are configured via group policies. This can be local group policies or active directory group policies. It can be confusing that the group policies for SRP and AppLocker can be configured on all versions of Windows, but do not apply on all versions of Windows. If AppLocker is not available, an event will be written to the AppLocker event logs mentioning that this Windows "SKU" (Stock-Keeping Unit) is not supported.




**AppLocker**

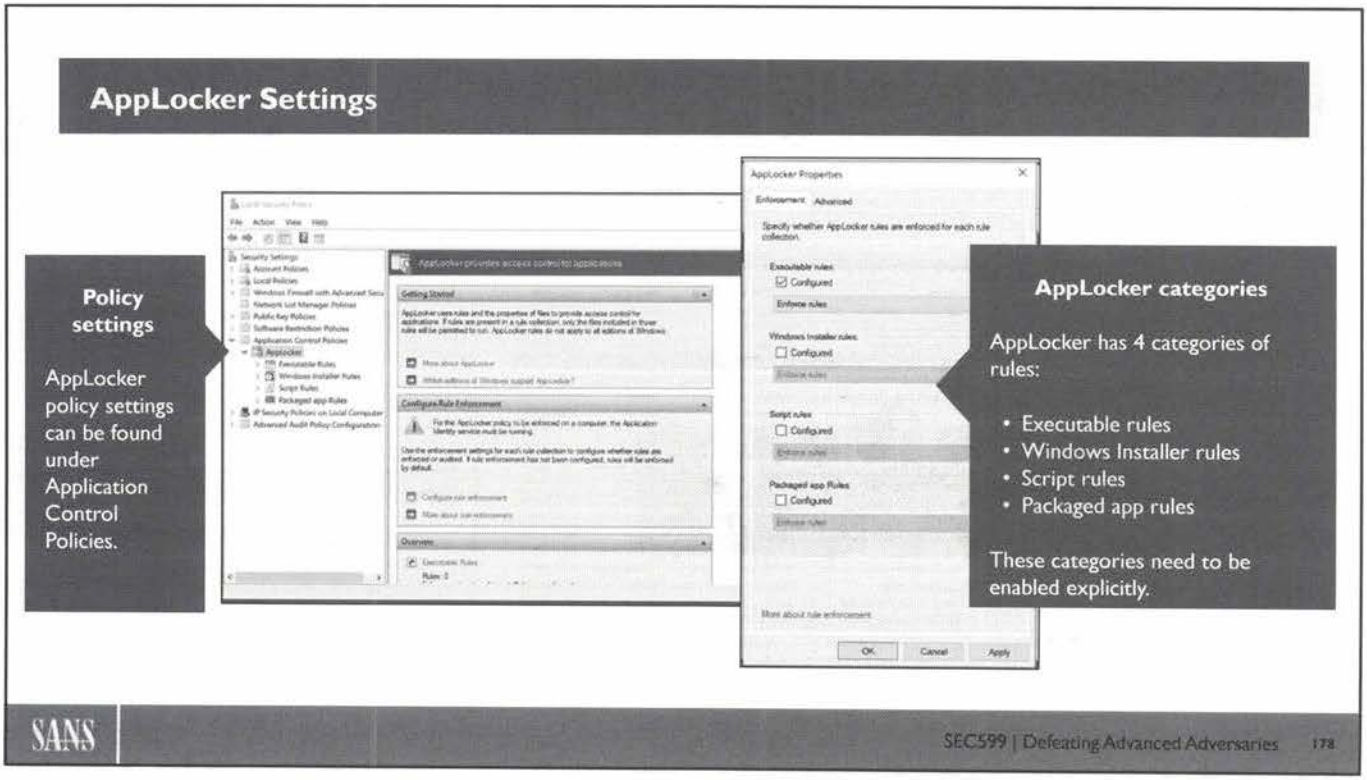
As AppLocker is the more recent implementation of application whitelisting technology, we will start our examples with AppLocker.

But before we can start creating rules, AppLocker needs some configuration.

First of all, AppLocker relies on a Windows service that is not running by default. The Application Identity service is a service that will support AppLocker identifying applications and decide what to block and allow.

If the Application Identity service is not running, AppLocker cannot identify applications and control execution.

We can start the service, and set its startup type to Automatic instead of Manual. In an enterprise, this is best done through active directory group policies. Remark that starting with Windows 10, the Application Identity service is a "protected service" and its settings (like Startup Type) cannot be controlled via the service manager. It has to be done with group policies.




**AppLocker Settings**

In these examples, we will use the local group policy editor. We decided this because the exercises will cover the active directory group policy editor.

In the local group policy editor, AppLocker policy settings can be found under Application Control Policies.

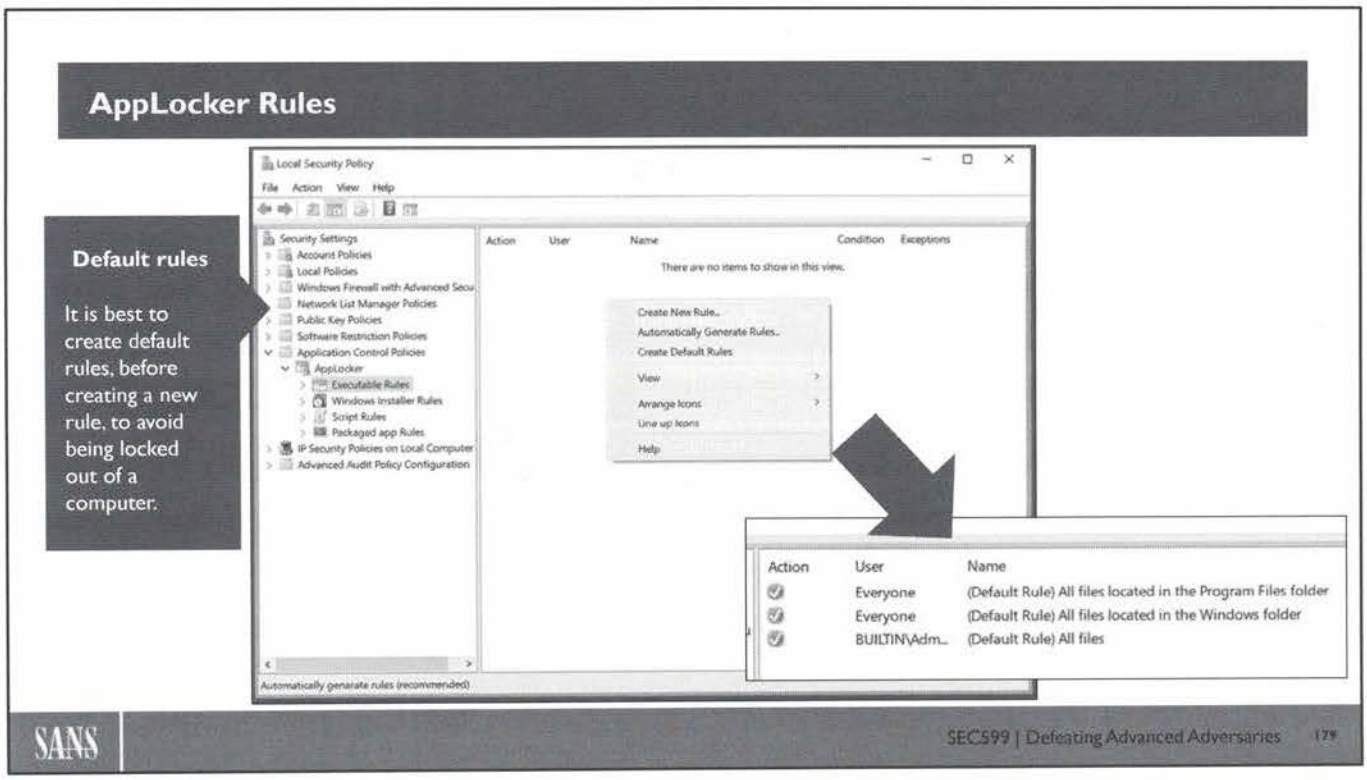
By default, no rules will be enforced by AppLocker. The AppLocker Properties group AppLocker rules by collections:

- Executable rules
- Windows Installer rules
- Script rules
- Packaged app rules

The last category of rules (packaged apps) control apps, which were introduced with Windows 8.

Rules in a category are only applied if the category is enforced. By default, no categories are enforced. In our example, we will enforce Executable rules. Remark that this can cause problems on Windows 8 and later: apps will not be allowed to run (on Windows 10, calculator is an app).

Besides enforcing rules, it is also possible to audit rules. This is useful to test a configuration: rules are applied but they are not enforced, thus applications are allowed to run. But an entry in the event log will mark what rules applied. This allows us to test our rules without running the risk of locking us out of our machine: if we make rules that are too restrictive, we can no longer run applications required to manage our machine, and we lose control over it.




**AppLocker Rules**

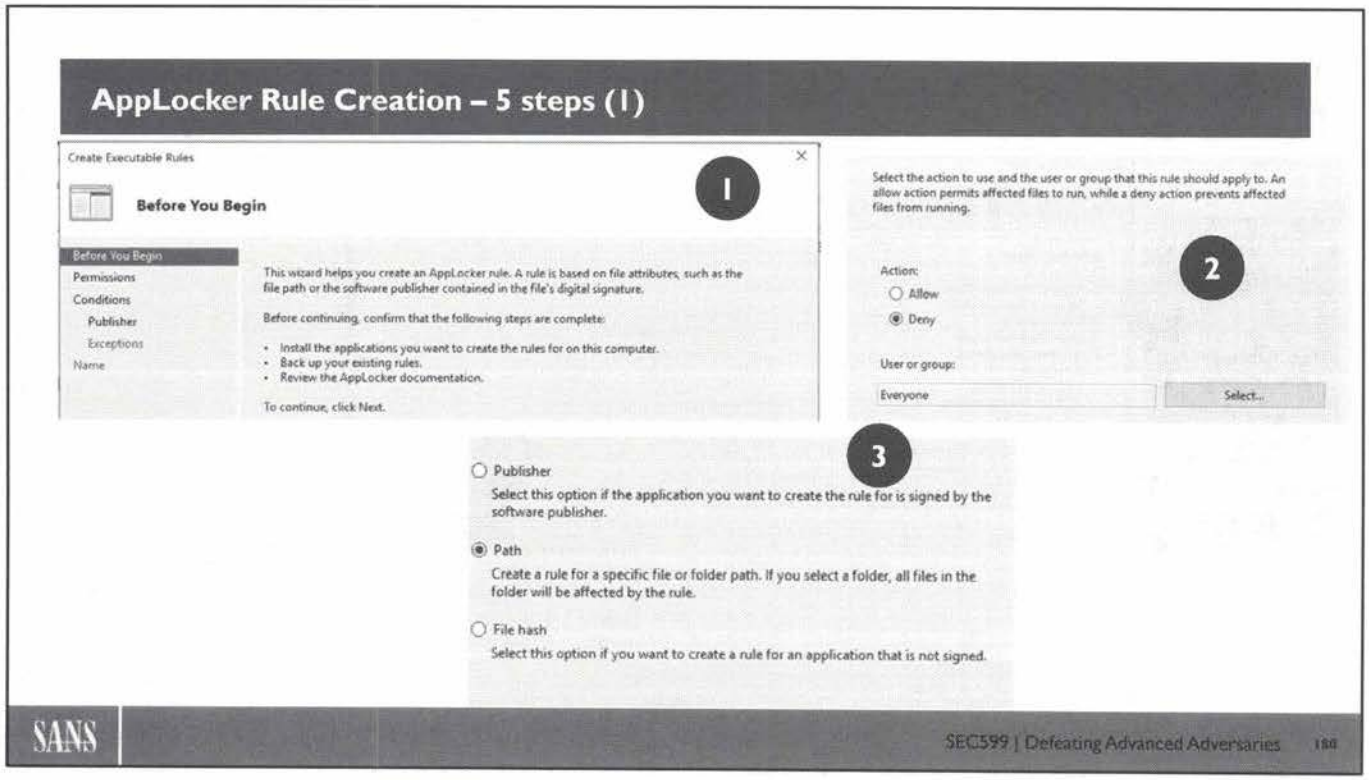
A last action to take before we can start creating our own rules is the creation of default rules. These have to be created by category of rules. It is not mandatory but recommended.

There are 3 default rules created for executable rules. These default rules can be created from the right-click menu, and they will also be suggested when we create our first rule (on the condition that the default rules do not exist).

The purpose of the default rules is to allow proper execution of existing Windows applications. There are 2 rules that apply to the group Everyone: they allow the execution of all executables in the Windows folders and in the Program Files folders. A third default rule applies to administrators only: administrators are allowed to execute all executables.

Remark that these default rules are only effective if proper control is applied to the Windows and Program Files folders. These folders must not be writable to users; otherwise, users can just copy executables into these folders and have them executed, thereby effectively bypassing AppLocker.

These rules also work under the assumption that normal users are not administrators. If your users need to be administrators on their machine, you will need to tune the default rules, as they will be allowed to run all executables.




**AppLocker Rule Creation – 5 steps (1)**
After fulfilling AppLocker's preconditions, we can start creating rules.

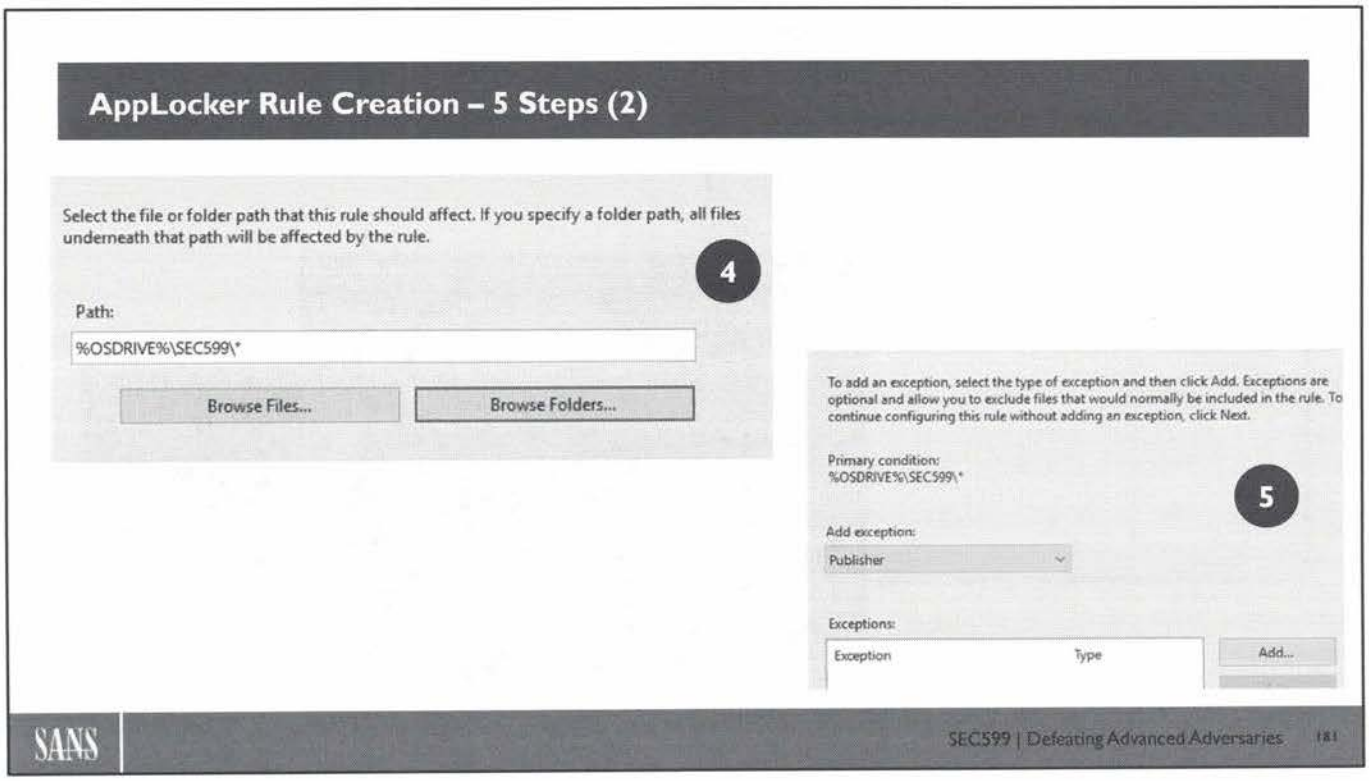
In the right-click menu in the Executable rules when can select "Create New Rule…".

This will display dialog number 1), presenting information. This dialog can be configured to be skipped.

In dialog number 2, we can decide if this is a rule to Allow or Deny applications (whitelisting or blacklisting). By default, the rule applies to group Everyone, but this can be changed.

In dialog number 3, we can select the criteria: the rule can identify executables by Publisher, Path or File hash. Publisher works with digital certificates and requires executables to be digitally signed. Path allows us to specify a filename or a folder (this includes subfolders), with or without wildcards. And File Hash allows us to select a file to calculate the cryptographic hash. All executables with the same cryptographic hash will be selected by this rule.

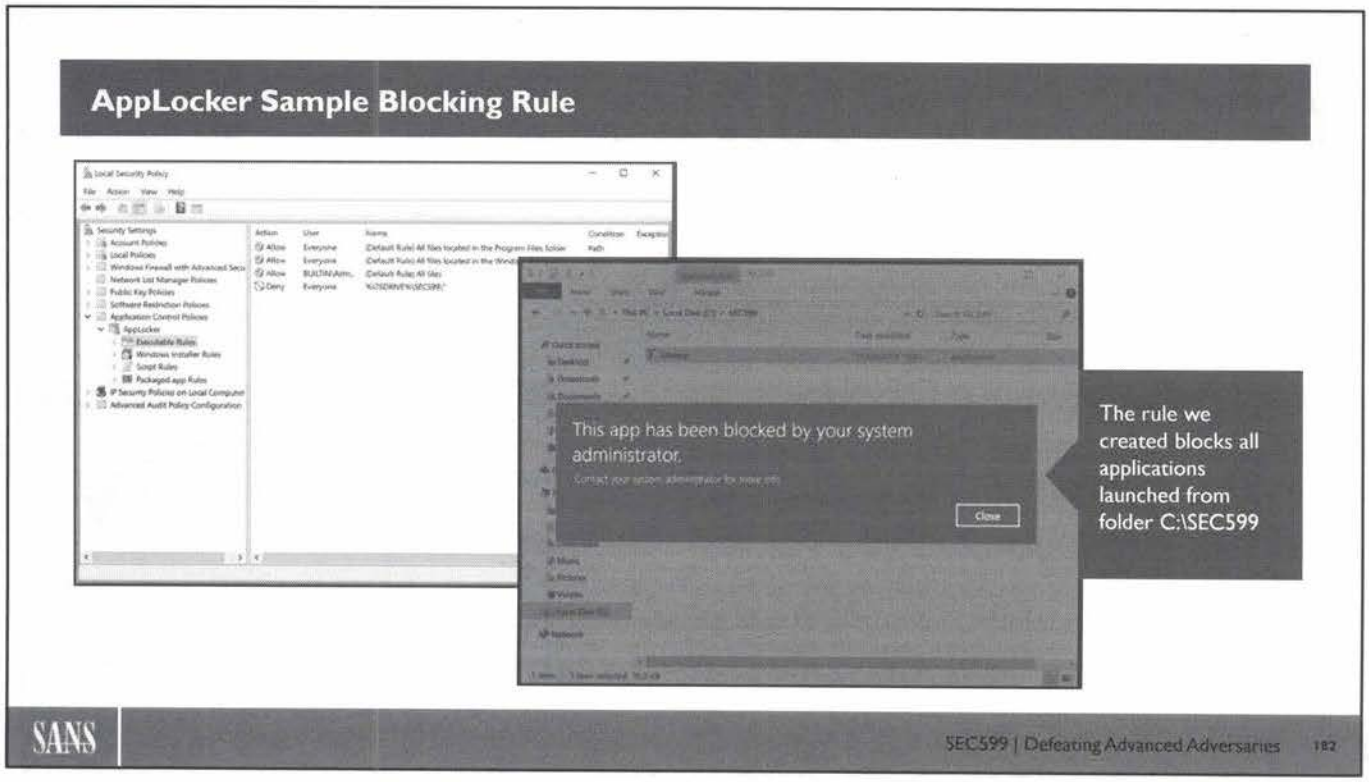
In our example, we select Path.




**AppLocker Rule Creation – 5 Steps (2)**

This brings us to dialog 4, where we can specify a filename or folder. We select folder C:\SEC599. Remark that the rule editor translated this to %OSDRIVE%\SEC599*. On our machine, variable %OSDRIVE% is equal to C:. AppLocker will use variables to make rules more general and applicable to a wider variety of Windows machines. For example, this rule will also work properly on a machine that has Windows installed on drive D:

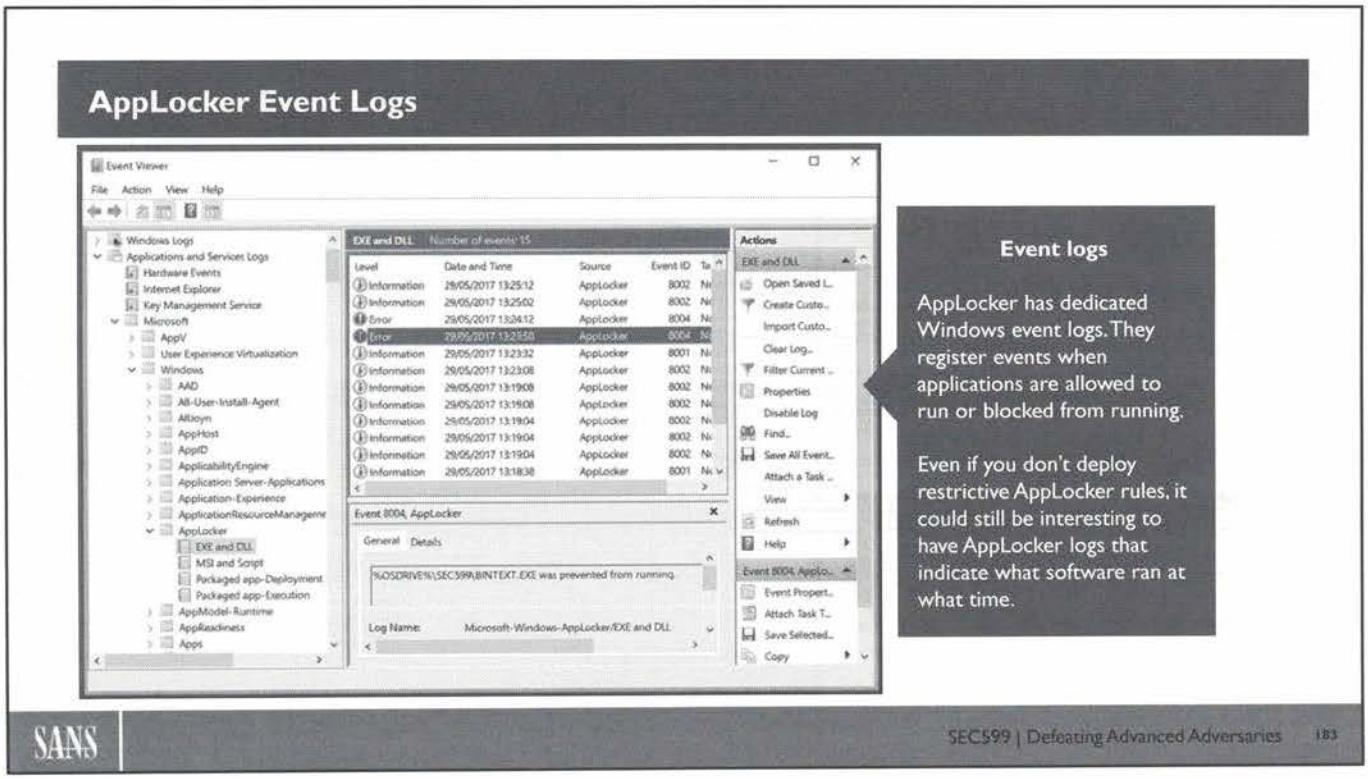
Finally, in dialog 5, we can add an exception to the rule.




**AppLocker Sample Blocking Rule**
Once the rule is created and applied, no user will be allowed to run executables from the C:\SEC599 folders.

When you are testing your own rules, be aware that rules have to be deployed first before they become effective. This can take some time, especially if you are deploying them using Group Policy Objects in Windows domain environments.

As an example, we try to execute executable bintext.exe in folder C:\SEC599. This is not allowed, and we are presented with a dialog box informing us that our system administrator prevented us from executing the executable. The dialog box presented in the slide above is for Windows 10. These dialog boxes vary per version of Windows.




**AppLocker Event Logs**
A very interesting feature of AppLocker is the dedicated Windows event logs.

In the event log viewer, under Applications and Service Logs / Microsoft / Windows / AppLocker we can find 4 event logs (3 on Windows 7): these are the event logs for the 4 categories of rules.

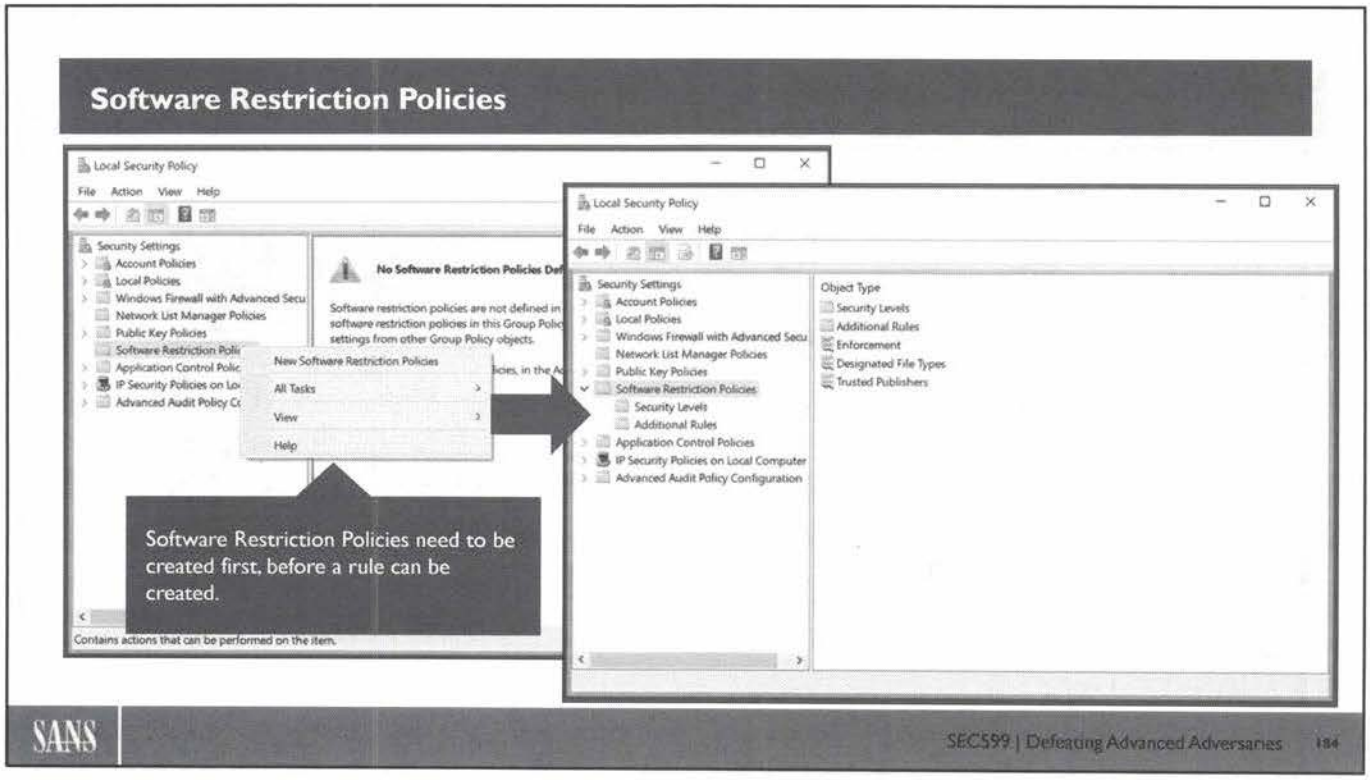
In the EXE and DLL log presented in the slide above, we selected the event that corresponds with the attempted execution of bintext.exe. As we can see from the details of the event, execution was blocked.

A similar event is created when an application is allowed to run.

These logs are very useful for logging and monitoring. By centralizing these logs and monitoring all blocked execution attempts, we can be informed of intrusion attempts.

The event log also contains events when policies are applied. When we create new rules to test, we watch the event log to detect deployment of our policies. When the policies have been deployed, we can start testing.

Remark that if a particular version of Windows (like Windows 10 Pro) does not support AppLocker, then an event will be created to inform us of this fact.



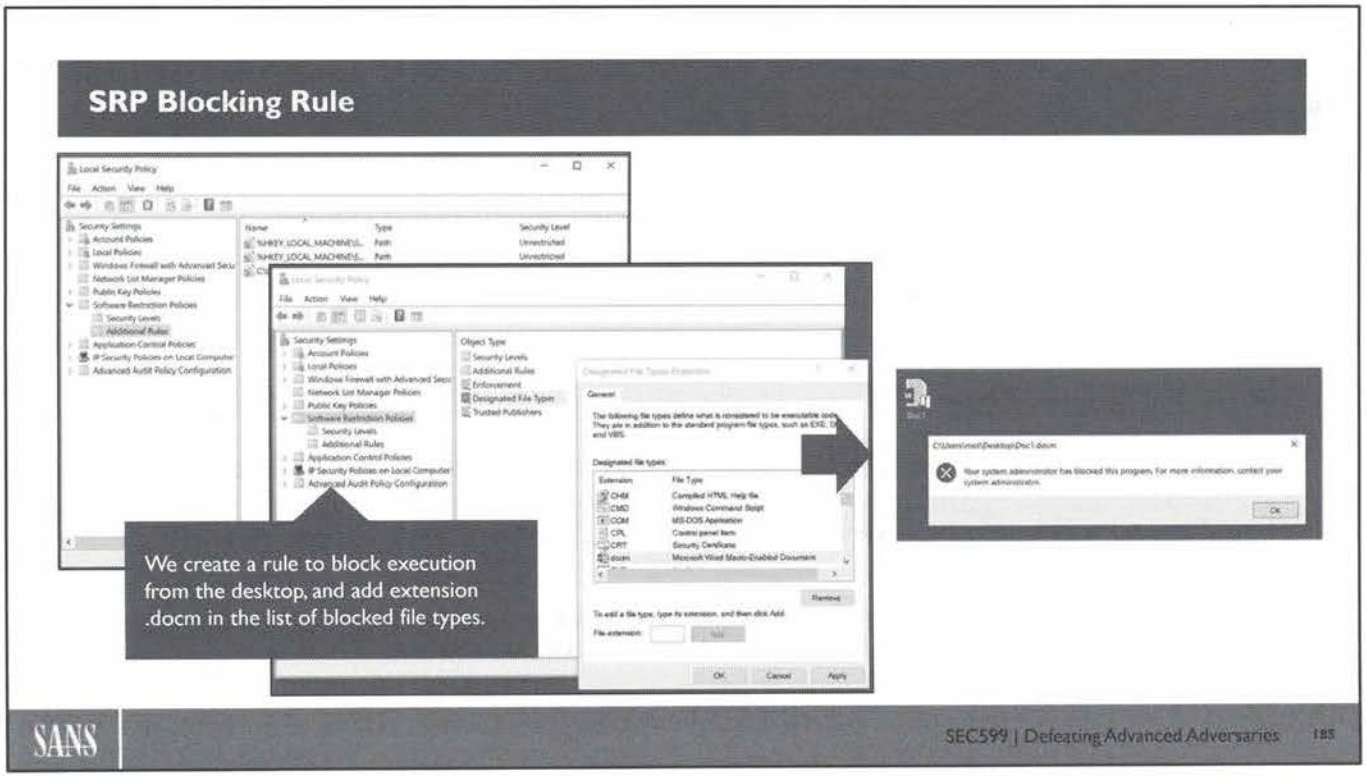

**Software Restriction Policies**

Although Software Restriction Policies are the older application whitelisting technology offered by Microsoft, and AppLocker should be the preferred technology to use, SRP has still some advantages.

Software Restriction Policies can be applied on versions of Windows that do not support AppLocker. So, in some cases, SRP can be your only option without having to resort to third–party solutions.

Software Restriction Policies also give us the option of specifying which type of files are to be considered executables. We will illustrate this in the next example.

Before we can start creating rules, we have to create the containers for the policies first, as shown in the slide above.




**SRP Blocking Rule**

As an example, we will show how we can block all Word Documents with macros using Software Restriction Policies.

First, we create a rule to Disallow all execution from our Windows Desktop: C:\Users\root\Desktop. Remark that this rule is not as flexible as AppLocker rules: it does not use variables, and thus only applies to one user.

This rule will prevent us from starting applications on our desktop. Remark that with Software Restriction Policies, the policies are applied at logon time. So, to apply this rule, we need to log out and log on again.

Next, in Designated File Types, we can specify the file types that are controlled by Software Restriction Policies. This is done by file extension (not by file content), and we add extension .docm to the list.

When we try to open document Doc1.docm, we get a Software Restriction Policy dialog preventing us from opening the document.

## Application Whitelisting – Bypass Techniques

As with any security control, a number of techniques exist that attempt to bypass implemented whitelisting controls. Some of the most popular ones include:

- PowerShell PE Injection (Prior to Windows 10)
- Using Installutil.exe
- Regsvr32.exe using scrobj.dll

A good overview of application whitelisting bypass techniques is being maintained by Oddvar Moe (https://github.com/api0cradle/UltimateAppLockerByPassList). We should keep an eye out on current techniques and adapt our defenses accordingly!

**Still, it's all about defense-in-depth! Application Whitelisting is a highly useful control, but it's not a silver bullet!**



Application Whitelisting – Bypass Techniques
As with any security control, a number of techniques exist that attempt to bypass implemented whitelisting controls. At the time of writing, some of the most popular (& successful) techniques include:

- PowerShell PE Injection (Prior to Windows 10)
- Using Installutil.exe
- Regsvr32.exe using scrobj.dll

A good overview of application whitelisting bypass techniques is being maintained by Oddvar Moe (https://github.com/api0cradle/UltimateAppLockerByPassList). We should keep an eye out for current techniques and adapt our defenses accordingly! Does that mean application whitelisting is broken? Of course not! It's all about defense-in-depth! Application Whitelisting is a highly useful control to have in your toolbox, but it's not a silver bullet!

# Course Roadmap

- Day 1: Knowing the Adversary, Knowing Yourself
- Day 2: Averting Payload Delivery
- **Day 3: Preventing Exploitation**
- Day 4: Avoiding Installation, Foiling Command & Control, & Thwarting Lateral Movement
- Day 5: Exfiltration, Cyber Deception & Incident Response
- Day 6: APT Defender Capstone

## SEC599.3

**Securing the Network**
- Network Access Control & 802.1X

**Securing Software**
- Software Development Lifecycle (SDL) & Threat Modeling
- Vulnerability Assessments
- Exercise: Authenticated Scans Using Nessus
- Patch Management
- Exploit Mitigation Techniques
- Exercise: Exploit Mitigation Using Compile-Time Controls
- Exploit Mitigation Techniques – Exploit Guard, EMET, & Others
- Exercise: Exploit Mitigation Using EMET & MalwareBytes

**Securing Endpoints**
- OS Hardening & Best Practices
- Endpoint Protection Solutions
- Application Whitelisting to Stop Payload Execution
- Exercise: Configuring AppLocker



This page intentionally left blank.

## Exercise – Configuring AppLocker

During this exercise, we will deploy a configuration for AppLocker that can be used to stop a malicious payload from executing. We will configure the AppLocker policy on the AD-level (domain) and push it to our clients using group policies.

High-level exercise steps:

1. Define the AppLocker application whitelisting configuration on domain-level
2. Push the configuration towards clients using group policies
3. Attempt to execute our malicious payloads to now see effective blocking of payloads
4. Test application whitelisting bypass technique



**Exercise – Configuring AppLocker**

The objective of the exercise is to analyze how AppLocker can be used to better protect our Windows environment / endpoints. During this exercise, we will deploy a configuration for AppLocker that can be used to stop a malicious payload from executing. We will configure the AppLocker policy on the AD-level (domain) and push it to our clients using group policies. The exercise consists of the following high-level steps:

1. Define the AppLocker application whitelisting configuration on domain-level
2. Push the configuration towards clients using group policies
3. Attempt to execute our malicious payloads to now see effective blocking of payloads
4. Illustrating an application whitelisting bypass technique

For additional guidance & details on the lab, please refer to the LODS workbook.

## Exercise – Configuring AppLocker – Conclusion

Application whitelisting tools such as AppLocker can be an excellent means of preventing malicious code execution if they are properly configured and setup

Although some organizations argue "it's difficult to implement", the author has seen many enterprise-grade organizations successfully implement application whitelisting thereby tremendously increasing their security posture!

*This does not mean that application whitelisting is your silver bullet:*

*Techniques bypassing whitelisting exist, but it's a strong control that will make life of the adversaries harder!*

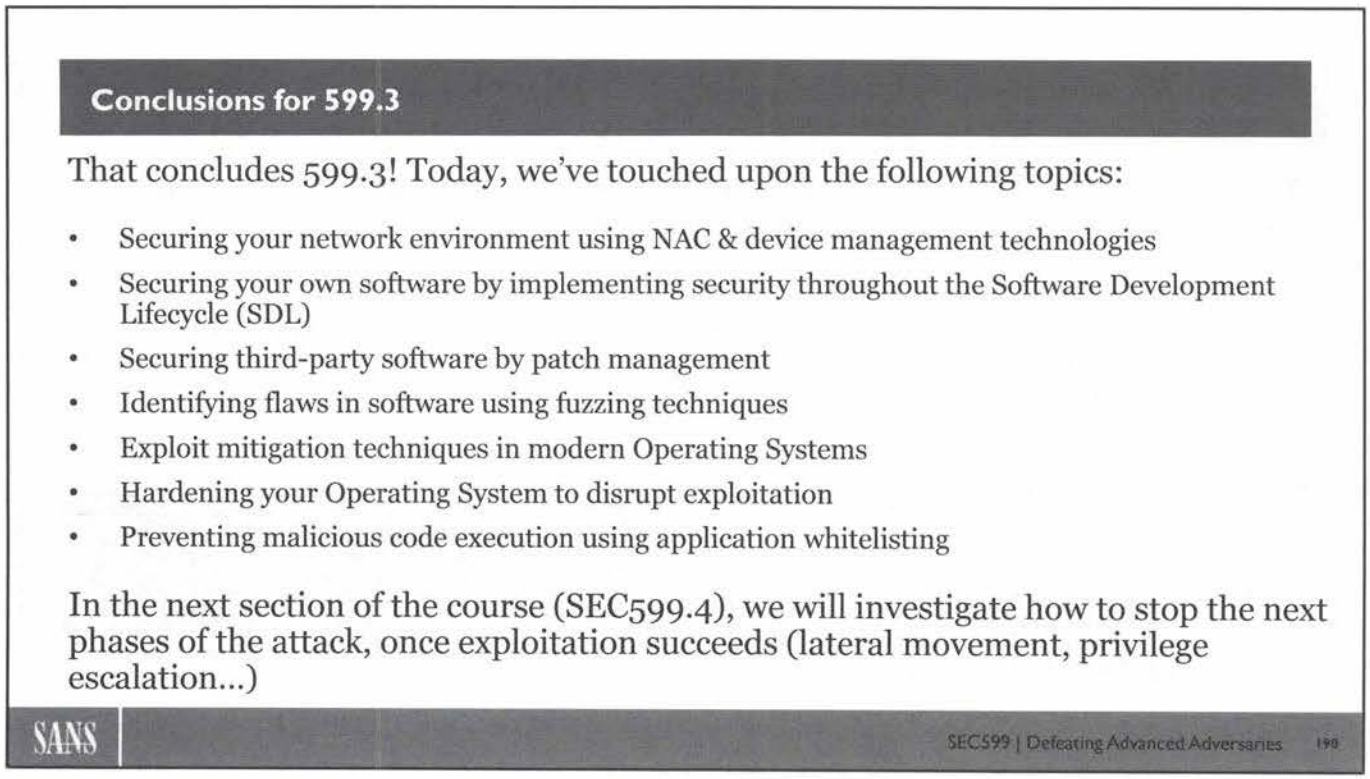



**Exercise – Configuring AppLocker – Conclusion**

In conclusion of this lab, we'd like to note that application whitelisting tools such as AppLocker can be an excellent means of preventing malicious code execution IF they are properly configured and setup. This is an important prerequisite: the technology needs to be properly implemented.

Although some organizations argue application whitelisting is "difficult to implement", the author has seen many enterprise-grade organizations successfully implement application whitelisting thereby tremendously increasing their security posture. Often, application whitelisting is affected by general IT & application "hygiene". If you cannot whitelist what applications your end-users are using, it might indicate you have an IT management issue which is broader than just "application whitelisting".

The effectiveness of application whitelisting does not mean it's a silver bullet against exploitation: Techniques bypassing whitelisting exist, but it's another strong control that will make life of the adversaries harder! And that's what it's all about, right? ☺

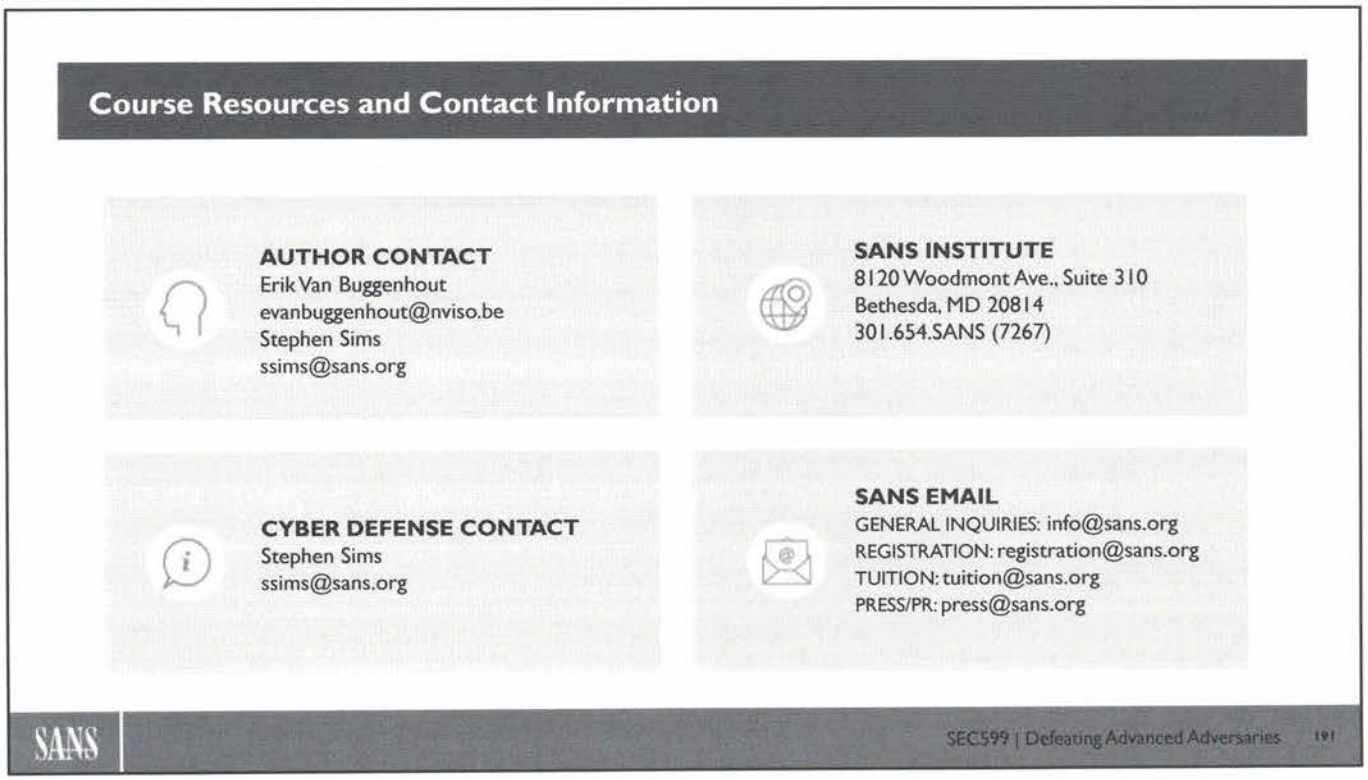



**Conclusions for 599.3**

So far 599.3! Today, we looked into how initial exploitation by adversaries can be prevented and detected. Amongst others, we touched upon the following topics:

- Securing your network environment using NAC & device management technologies
- Securing your own software by implementing security throughout the Software Development Lifecycle (SDL)
- Securing third-party software by patch management
- Identifying flaws in software using fuzzing techniques
- Exploit mitigation techniques in modern Operating Systems
- Hardening your Operating System to disrupt exploitation
- Preventing malicious code execution using application whitelisting

In the next section of the course (SEC599.4), we will investigate how to stop the next phases of the attack, once exploitation succeeds (lateral movement, privilege escalation…)




This page intentionally left blank.